Aproximação e Compartilhamento de Custos em Projeto de Redes

André Luís Vignatti

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Aproximação e Compartilhamento de Custos em Projeto de Redes

André Luís Vignatti[1]

14 de fevereiro de 2006

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa
  Instituto de Computação, Unicamp (Orientador)
- Prof. Dr. Orlando Lee
  Instituto de Computação, Unicamp
- Prof. Dr. Eduardo Sany Laber
  Departamento de Informática, PUC-Rio
- Prof. Dra. Yoshiko Wakabayashi
  Instituto de Matemática e Estatística, USP (Suplente)

[1] **Auxílio financeiro da CNPQ processo NUMERO 131583/2004-2**

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Vignatti, André Luís

V683a Aproximação e compartilhamento de custos em projeto de redes / André Luís Vignatti -- Campinas, [S.P. :s.n.], 2006.

Orientador : Flávio Keidi Miyazawa

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Teoria da computação. 2. Algoritmos – Programação linear. 3. Otimização combinatória. I. Miyazawa, Flávio Keidi. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Approximation and cost-sharing in network design

Palavras-chave em inglês (Keywords): 1. Computer theory. 2. Algorithms – Linear programming. 3. Combinatorial optimization.

Área de concentração: Teoria da Computação

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Flavio Keidi Miyazawa (IC-UNICAMP)
Prof. Dr. Orlando Lee (IC-UNICAMP)
Prof. Dr. Eduardo Sany Laber (Dimf-PUC-RIO)

Data da defesa: 14/03/2006

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 14 de março de 2006, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Eduardo Sany Laber
PUC-RJ.

Prof. Dr. Orlando Lee
IC / UNICAMP.

Prof. Dr.Flávio Keidi Miyazawa
IC / UNICAMP.

Aproximação e Compartilhamento de Custos em Projeto de Redes

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por André Luís Vignatti e aprovada pela Banca Examinadora.

Campinas, 14 de março de 2006.

Prof. Dr. Flávio Keidi Miyazawa
Instituto de Computação, Unicamp
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Resumo

Neste trabalho estudamos a interação entre duas áreas: otimização combinatória e compartilhamento de custos (cost-sharing), que é a arte de dividir os custos associados à construção e manutenção de uma solução a qual um grupo de usuários é beneficiado. Apresentamos algoritmos para problemas de projeto de redes, tendo como objetivo principal os problemas "Connected Facility Location" e "Rent-or-Buy". Estes dois problemas são NP-difíceis, pois têm como caso particular o problema da árvore mínima de Steiner, que também é NP-difícil. Na primeira parte do trabalho, temos a seguinte questão como motivação: "Como projetar uma boa rede, ou seja, uma rede que satisfaça todas as propriedades do problema e ao mesmo tempo minimize o custo de construção desta rede?" É nesta parte que os algoritmos de aproximação entram em ação. Uma vez que esse custo for determinado, na segunda parte do trabalho, uma outra questão surge: "Como dividir esse custo entre todos os usuários que participam da rede de uma maneira "justa"? " Nesta parte, usaremos o compartilhamento de custos juntamente com as técnicas de algoritmos de aproximação para responder a essa questão.

Abstract

We consider the interplay of two areas: combinatorial optimization and cost-sharing in network design problems. In the first, we are interested to find a solution with small cost. In the second we would like to share the solution cost between its users. We present algorithms for the problems "Connected Facility Location" and "Rent-or-Buy". These two problems are NP-hard, since they have as a particular case the minimum Steiner tree problem, which is a known NP-hard problem. In the first part of this work, we have the following question as motivation: "how to design a good network, i.e., one that satisfies all problem requirements and minimize the overall network construction cost?" In this part, approximation algorithms takes action. Once this cost is determinated, in the second part of the work, another question arises: "How to distribute this cost among all users that participate in the network in a "fair" way?" In this part, we will use cost-sharing together with approximation algorithms techniques to answer this question.

ao meu irmão Tiago
aos meus pais Elusa e Olir
e à memória de minha avó Terezinha

Agradecimentos

Parece que finalmente consegui! E não vou economizar nos agradecimentos, ao contrário, faço-os com capricho, já que todos os que olham a dissertação, sempre lêem esta seção.

Dado que a atribuição de um pai e uma mãe a uma criança que vai nascer é um evento aleatório, eu ganhei na loteria: tenho muita sorte de ser filho da minha mãe Elusa (o nome dela vem antes, senão ela iria brigar comigo) e do meu pai Olir, a eles meus mais sinceros agradecimentos. Não menos importante, devo agradecer também a minha vó Terezinha, que já se foi, mas ainda está aqui comigo, e meu irmão Tiago, que é o melhor irmão-companheiro-de-agitos-de-conversas-de-banda-e-de-gostar-de-ser-computeiro!

Quando chegamos na pós-graduação, nem sempre temos muita certeza sobre qual dos pequeníssimos ramos das áreas do conhecimento iremos escolher para nos especializar, o que pode nos causar arrependimentos futuros. Não bastasse isso, cheguei na UNICAMP sem conhecer um único professor! Assim, meu orientador, o Flávio, foi um cara que caiu do céu. Sempre muito paciente, preocupado, e nunca de mau humor. Devo agradecê-lo por tudo isso, e por deixar por minha conta a escolha dos problemas a serem estudados, assim como ter confiado e me dado liberdade de extrapolar um pouco em alguns assuntos que eram novidade tanto pra mim quanto pra ele. Essa dissertação é o resultado dessa excelente orientação.

Não devo esquecer também dos grandes professores que tive na graduação na UFPR. Agradeço o prof. Elias, que abriu algumas portas para mim, oferecendo-me uma Iniciação Científica e participando junto com ele da organização de um congresso. Agradeço também o professor Hélio, excelente professor que também me incentivou bastante e deu dicas quanto à vinda para Campinas. Agradeço também aos professores Alexandre, Castilho, Roberto. Além disso, devo agradecimentos especiais ao professor Jair Donadelli, que foi meu orientador de trabalho de conclusão de curso e orientador de iniciação científica, me explicando com paciência muita das coisas que sei hoje sobre a computação teórica. Devo também agradecê-lo pelos bate-papos furados sempre engraçados, assim como bate-papos sérios que com certeza influenciaram minha decisão por fazer pós-graduação. Além disso, foi ele quem insistiu para que eu tentasse mestrado fora da UFPR, se não fosse por ele, talvez não tivesse vindo para Campinas.

Tem um cara que me acompanha desde o ensino fundamental, naquela época, ainda estudávamos em salas diferentes. Já no ensino médio, estudamos na mesma sala. Mais tarde passamos no vestibular para o mesmo curso, fizemos todas as disciplinas juntos e também grande parte dos trabalhos da faculdade. Nos formamos na mesma data. Resolvemos tentar a sorte fazendo pós-graduação. Passamos na seleção da pós nas mesmas universidades. Decidimos os dois estudar em Campinas. Desde então, dormimos no mesmo quarto, mas uma coisa a gente não tem em comum: a cama! Esse é o meu grande amigo Luiz Fernando Bittencourt, mais conhecido como bitE (com E maiúsculo, para acentuar o sotaque de Curitiba). Agradeço a ele por ter me agüentado por todo esse tempo, em especial nesses dois últimos anos em Campinas. Além disso, não posso esquecer seus grandes ensinamentos, como por exemplo, utilizar o abridor de latas e aprender a dar laço que não seja laço no tênis. Tive que fazer mestrado pra aprender isso!

Agradeço à Maíra, minha namorada, por ela existir e ter me avisado de sua existência enviando aquele scrap no orkut: se não fosse isso, nunca iríamos nos conhecer. Agradeço a ela por todos os momentos, conversas, carinhos, dedicação e compreensão. Com certeza, ela teve fundamental importância em tudo que o fiz nesse último ano. Também gostaria de agradecer a Marli, a minha sogrona, que se preocupa bastante comigo, compra bolo no meu aniversário e faz quiabo sem baba especialmente pra mim. Tudo isso fez com que minha aventura solitária do mestrado tivesse um clima mais "familiar", já que a minha família está longe.

Agradeço à cidade de Campinas por ser uma cidade sem muitas opções de lazer onde tudo é caro, fazendo com que, diante da fome mundial, da miséria, das guerras e etc, eu me concentrasse na dissertação que eu tinha que terminar.

Agradeço aos piás "camarões" de Curitiba; Eduardo, Rafael, Richard, Wagner, Gustavo, Tim (vulgo Bruno), Batata e Luiz Lançoni, que cresceram junto comigo e são os melhores amigos que alguém pode ter. Além disso, exposto a minha praticamente inexistente vida social em Campinas, fazem com que todas as minhas idas à Curitiba sejam bem divertidas, com muita festa, rock'n roll, bebida e papo-furado.

Agradeço à minha amiga Suelen, por todos os bons momentos que passamos juntos e por nunca ter me desencorajado a vir pra Campinas (pelo contrário, ela deu o maior apoio).

Devo também dizer que não sou, de forma alguma, o mesmo André de dois anos atrás, quando iniciei o mestrado. Além de toda a experiência de morar sozinho, ganhei também ao conhecer vários novos amigos. A primeira guria que me vem a cabeça é a Sheila. Agradeço a ela por toda a amizade, companheirismo, festas (nos conhecemos em uma), os jantares na sua república, as conversas (inclusive as fofocas), etc. Com certeza, ela é uma computeira bem fora do padrão (no caso, pra melhor). Agradeço os "fio do Frávio", Pará (piázão bom de bola), Carlos Eduardo (o cara mais camarão do IC), Evandro e

Luiz Augusto. Todos muito gente boa e muito crânio! Agradeço também os colegas do laboratório LAD (ou LUG? Ou LOCO?), ao pessoal da cerveja no StarClean e a todos que conheci nesses dois anos que são do IC e que são gente boa. Agradeço também os vizinhos que conheci no condomínio onde moro.

A seguir, agradeço às coisas que não fazem parte dos agradecimentos principais, mas que não merecem ficar de fora, pois de alguma forma me acompanharam ou participaram significativamente da minha vida de mestrando. Agradeço às festas do DCE, IA e IFCH, assim como gostaria de agradecer às pessoas que gastam seus tempos preciosos para organizar essas festinhas sempre muito legais. Agradeço também aos amigos e amigas que conheci nestas festas. Agradeço ao Frank Zappa, Pink Floyd e a todos os grande músicos que me inspiraram e continuam me inspirando, com certeza não vivo sem suas músicas. Agradeço também à minha ex-banda João Lanterna, que continua firme e forte lá em Curitiba fazendo o melhor som que conheço! Agradeço aos organizadores do festival Psicodália, que nos proporcionam a cada festival um mix único entre música/cultura/natureza/festa, inclusive o festival excelente e surreal que ocorreu no carnaval.

Por fim, gostaria de agradecer o apoio financeiro da CNPq e a toda população brasileira que contribui indiretamente para sustentar a universidade pública no Brasil.

Poeta itinerante e peregrino,
pelas ruas do mundo,
arrasto o meu destino
Mundo? Uma aldeia de nome tupi,
um monstro com nome de santo,
Curitiba, São Paulo,
com vocês me deito,
com algo me levanto.
Vocês aí parados
a mesma vida de sempre,
como vos invejo e vos desprezo,
voz de nós, voz dos meus avós,
prazos, prêmios, praças, preços,
chove sobre mim
a chuva que eu mereço.
Invoco forças poderosas.
Quando vou poder
transformar minhas ruínas em rosas ?

(Paulo Leminski - Poeta Itinerante - 1988)

Sumário

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Esta dissertação é sobre a interação entre duas áreas: otimização combinatória e compartilhamento de custos (cost-sharing), que é a arte de dividir os custos associados à construção e manutenção de uma solução a qual um grupo de usuários é beneficiado. Neste trabalho apresentamos algoritmos para problemas de projeto de redes, tendo como objetivo principal os problemas "Connected Facility Location" e "Rent-or-Buy". Estes dois problemas são NP-difíceis, pois têm como caso particular o problema da árvore de Steiner de peso mínimo, que também é NP-difícil [23]. Na primeira parte do trabalho, temos a seguinte questão como motivação:"Como projetar uma boa rede, ou seja, uma rede que satisfaça todas as propriedades do problema e ao mesmo tempo minimize o custo de construção desta rede?" É nesta parte que os algoritmos de aproximação entram em ação. Uma vez que esse custo for determinado, na segunda parte do trabalho, uma outra questão surge: "Como dividir esse custo entre todos os usuários que participam da rede de uma maneira "justa"?" Nesta parte, usaremos o compartilhamento de custos juntamente com as técnicas de algoritmos de aproximação para responder a essa questão.

Algoritmos de aproximação é um ramo da otimização combinatória que está preocupado em encontrar soluções próximas da solução ótima. Falaremos depois sobre algoritmos de aproximação, antes vamos explicar o que é otimização combinatória.

1.1 Otimização combinatória

Problemas de otimização, em sua forma geral, têm como objetivo maximizar ou minimizar uma função definida sobre um certo domínio. A teoria clássica de otimização trata do caso em que o domínio é infinito. Já no caso dos chamados problemas de otimização combinatória, o domínio é tipicamente finito; além disso, em geral é fácil listar os seus elementos e também testar se um dado elemento pertence a esse domínio. Ainda assim, a idéia ingênua de testar todos os elementos deste domínio na busca pelo melhor mostra-se

inviável na prática, mesmo para instâncias de tamanho moderado.

Nesta dissertação estamos considerando a hipótese de P≠NP. Assim, problemas NP-difíceis, dentre eles o Connected Facility Location e o Rent-or-Buy, são impossíveis de serem resolvidos por algoritmos eficientes, i.e., algoritmos que executem em ordem de tempo polinomial no tamanho da entrada em computadores baseados na máquina de Turing tradicional. No entanto, existem várias técnicas alternativas, cada qual compreendendo uma sub-área da otimização combinatória, para tentar resolver tais problemas de forma satisfatória, onde cada técnica possui vantagens e desvantagens com relação às outras. Dentre as técnicas utilizadas, podemos citar o uso de heurísticas, programação inteira, métodos híbridos, redes neurais, algoritmos genéticos e outros.

1.1.1 Algoritmos de aproximação

A técnica de otimização combinatória que estaremos interessados neste texto são os *algoritmos de aproximação*. Nesta técnica, sacrificamos a otimalidade da solução em troca da garantia de uma solução aproximada computável eficientemente. Certamente, o interesse é, apesar de sacrificar a otimalidade, fazê-lo de forma que ainda possamos dar boas garantias sobre o valor da solução obtida, procurando ganhar o máximo em termos de eficiência computacional. Em linhas gerais, algoritmos de aproximação são algoritmos que não necessariamente produzem uma solução ótima, mas soluções que estão dentro de um certo fator da solução ótima.

1.2 Compartilhamento de custos

Uma forma mais realista de tratar problemas não está somente preocupada com o custo da solução, como é o caso da otimização combinatória. Além disso, temos usuários que participam da solução que devem ser cobrados com uma parcela do custo da solução. No entanto, podemos dividir o custo da solução de várias maneiras diferentes. Algumas dessas maneiras privilegiam um certo grupo de usuários, mas podem trazer consequências negativas a outro grupo. O objetivo do compartilhamento de custos é realizar a divisão dos custos da solução encontrada de forma que esta divisão seja justa para todos os usuários, encorajando a cooperação entre os usuários.

O compartilhamente de custos tem origem na teoria dos jogos, mais especificadamente, a teoria dos jogos cooperativos. Como esta é uma área muito grande do conhecimento, muitos dos estudos realizados sobre o assunto serão somente enunciados como teoremas, sendo referenciados na bibliografia. A idéia de juntar algoritmos de aproximação com compartilhamento de custos é um assunto bastante recente e ativo, com resultados novos aparecendo regularmente.

Estamos interessados em encontrar funções de compartilhamento de custo que capturem a noção de "cross-monotonicity"[1], que diz que o custo cobrado de cada usuário não deve aumentar se adicionarmos usuários ao sistema, e da mesma forma, não deve diminuir quando usuários saem do sistema.

1.3 Projeto de Redes

Os problemas de projeto de redes são modelados como problemas em grafos que contêm características inerentes ao problema original, como custos nas arestas e vértices, arestas direcionadas, etc. O progresso da pesquisa e conseqüentemente a publicação de artigos científicos que tratam de problemas de projeto de redes tem sido constante nos últimos anos. De fato, o interesse por esses problemas é muito grande devido a atual tecnologia que possibilita a criação de projeto físico de redes com milhares de nodos. Estes problemas têm várias aplicações, principalmente em projeto de redes de telecomunicações, distribuição de caches e roteadores, aglomeração de dados em tráfego e projeto de localização de facilidades (como pontos de correio, corpo de bombeiros, bancas de revista, etc).

Neste trabalho estamos interessados em investigar os problemas *Connected Facility Location* (CFL), assim como variantes e problemas relacionados, como o problema *Rent-or-Buy*, que é um caso particular do CFL. Como objetivo, queremos aplicar técnicas de algoritmos de aproximação e compartilhamento de custos nestes problemas. No CFL, temos um grafo conexo, um conjunto de facilidades que podem ser abertas e um conjunto de demandas que devem ser conectadas às facilidades. O objetivo é abrir facilidades, conectando todas as demandas nas facilidades abertas e definir uma árvore de Steiner sobre as facilidades que foram abertas tal que o custo total seja minimizado. Em outras palavras, o CFL é o problema clássico *Facility Location* com a restrição adicional de que todas as facilidades abertas devem estar conectadas. Iremos fazer as definições formais destes problemas no Capítulo 2.

1.4 Organização do texto

No Capítulo 2, iremos fazer uma breve introdução aos conceitos preliminares que serão úteis nesta dissertação. Explicaremos o que são algoritmos de aproximação e definiremos formalmente os problemas os quais iremos tratar durante o texto. Adiaremos, no entanto,

[1] Ao longo desta dissertação, alguns termos técnicos foram deixados propositalmente em língua inglesa. O tema de compartilhamento de custo ainda é um assunto muito recente, e não há um consenso sobre a tradução da maioria dos termos técnicos.

a explicação dos conceitos preliminares sobre compartilhamento de custos para o Capítulo 5.

Nos Capítulos 3 e 4 iremos estudar duas técnicas distintas para obter algoritmos de aproximação. A técnica *primal-dual* será vista no Capítulo 3, onde obtemos algoritmos de aproximação para os problemas Rent-or-Buy e Connected Facility Location. No Capítulo 4 será apresentado um *algoritmo probabilístico* para o Rent-or-Buy, e usaremos técnicas de *métodos probabilísticos* para demonstrar o fator de aproximação. A grande vantagem deste método com relação ao método primal-dual é a simplidade do algoritmo e da análise respectiva. Por outro lado, temos como ponto negativo o fato do algoritmo apresentado não ter uma desaleatorização eficiente.

Nosso objetivo nesse trabalho não é apresentar todas as técnicas existentes para resolução desses problemas. Existem inúmeras técnicas para obter algoritmos de aproximação, escolhemos aqui as que exemplificam de forma satisfatória uma determinada técnica ou que ilustram algum fato que julgamos importante. De modo particular para os problemas Connected Facility Location e Rent-or-Buy, uma outra técnica foi utilizada em 2001 por Gupta et al. [11], que consiste do arredondamento de um programa linear resolvido em sua forma relaxada (fracionária). Os fatores de aproximação são piores dos que os obtidos pelos artigos subseqüentes, no entanto, este foi o primeiro trabalho a conseguir fatores de aproximação para esse problemas.

A segunda parte do trabalho começa a partir do Capítulo 5, o qual é inteiro dedicado a uma introdução sobre o assunto de compartilhamento de custos. Faremos definições necessárias, explicaremos os conceitos e resolveremos um problema simples para exemplificar o que está por vir nos próximos capítulos.

No Capítulo 6 utilizaremos a técnica primal-dual, já vista anteriormente, para obter funções de compartilhamento de custo. A técnica apresentada neste capítulo é uma técnica genérica que pode ser aplicada à outros problemas. Iremos apresentar funções de compartilhamento de custo para os problemas Facility Location, Rent-or-Buy e Connected Facility Location.

No Capítulo 7 utilizamos novamente a técnica probabilística, mas agora com o objetivo de obter um bom método de compartilhamento de custo.

Para concluir a dissertação, no Capítulo 8 apresentamos uma desaleatorização do algoritmo utilizado nos Capítulos 4 e 7, garantindo as propriedades de cost-sharing e um bom fator de aproximação. São usados conceitos sofisticados em métodos probabilísticos para conseguir uma desaleatorização do algoritmo apresentado.

Finalmente, apresentamos nossas conclusões e considerações finais no Capítulo 9.

No Apêndice A, apresentamos algumas definições básicas de conceitos que são utilizados no Capítulo 8, bem como a construção explícita de espaços amostrais pequenos, utilizado também neste mesmo capítulo.

Parte I

Algoritmos de aproximação

Capítulo 2

Preliminares

Este capítulo contém, de forma resumida, definições e noções básicas que serão necessárias no decorrer da leitura do trabalho. Apresentamos definições básicas sobre algoritmos de aproximação, assim como definições relacionadas sobre o tema. Além disso, discutimos brevemente algumas técnicas usadas no desenvolvimento de algoritmos aproximados. Também apresentamos os problemas de projeto de redes que iremos considerar neste trabalho, mostrando suas definições e discutindo algumas aplicações.

2.1 Algoritmos de aproximação

Para resolver os problemas propostos neste trabalho, utilizaremos a abordagem de *algoritmos de aproximação*. Abaixo, uma explicação sobre o que são os algoritmos de aproximação.

Dado um algoritmo A para um problema de minimização, se I é uma instância para este problema, vamos denotar por $A(I)$ o valor da solução devolvida pelo algoritmo A aplicado à instância I e vamos denotar por $\mathrm{OPT}(I)$ o valor correspondente para uma solução ótima de I. Diremos que um algoritmo tem um fator de aproximação α, ou é α-aproximado, se $A(I)/\mathrm{OPT}(I) \leq \alpha$, para toda instância I. No caso dos algoritmos probabilísticos, consideramos a desigualdade $E[A(I)]/\mathrm{OPT}(I) \leq \alpha$, onde a esperança $E[A(I)]$ é tomada sobre todas as escolhas aleatórias feitas pelo algoritmo. Na presente dissertação, α é um valor que não depende de I, embora em geral α possa ser uma função de I. Dizemos que um algoritmo polinomial A_ϵ, $\epsilon > 0$, para um problema de minimização, é um esquema de aproximação se o algoritmo tem um fator de aproximação $(1 + \epsilon)$, para todo $\epsilon > 0$. Neste trabalho, não apresentaremos nenhum esquema de aproximação. De fato, há resultados fortes que indicam a ausência de algoritmos deste tipo para os problemas que iremos considerar.

Nos últimos anos, surgiram várias técnicas de caráter geral para o desenvolvimento

de algoritmos de aproximação. Algumas destas são: *arredondamento de soluções via programação linear, dualidade em programação linear e método primal-dual, algoritmos probabilísticos e sua desaleatorização, relaxação lagrangeana, provas verificáveis probabilisticamente e a impossibilidade de aproximações,* dentre outras (veja [4, 14, 35, 36]).

Uma estratégia comum para tratar problemas de otimização combinatória, é formular o problema através de um sistema de programação linear inteira, resolver a relaxação linear deste (em tempo polinomial) e em seguida usar algum método para obter soluções inteiras, não necessariamente ótimas, através da solução da relaxação linear. O método de programação linear tem sido bastante usado no desenvolvimento de algoritmos aproximados, através de diversas maneiras: fazendo uso de arredondamentos das soluções fracionárias do programa linear; resolvendo o sistema dual do programa linear, em vez do primal, e em seguida obtendo uma solução com base nas variáveis duais; e fazendo uso de uma generalização do método primal-dual, que tem sido usado para obter diversos algoritmos combinatórios rápidos (sem uso da resolução de um programa linear).

Já no caso de algoritmos probabilísticos, o algoritmo contém passos que dependem de uma seqüência de bits aleatórios, dados na entrada. Neste caso a análise da solução gerada pelo algoritmo é feita com base no valor esperado da solução. É interessante observar que apesar do modelo parecer restrito, a maioria dos algoritmos probabilísticos pode ser desaleatorizada, através do método das esperanças condicionais, tornando-os algoritmos determinísticos (veja [26]). A versão probabilística é em geral mais simples de se implementar e mais fácil de se analisar que a correspondente versão determinística. Além disso, muitos dos algoritmos de aproximação combinam o uso de técnicas de programação linear com técnicas usadas em algoritmos probabilísticos, considerando o valor das variáveis obtidas pela relaxação linear como probabilidades.

A técnica de relaxação lagrangeana é bastante usada na área de otimização combinatória mas apenas recentemente tem sido considerada na obtenção de algoritmos de aproximação. Esta técnica é bem conhecida pela obtenção, em geral rápida, de limitantes para a solução ótima.

2.2 Os problemas Rent-or-Buy e Connected Facility Location

Antes de apresentarmos os problemas, vamos fazer uma observação que será utilizada em todos os problema deste documento. Em muitas situações práticas de projeto de redes, é sempre mais econômico ir diretamente de um vértice u a um vértice w, sem que tivéssemos que desviar por algum outro vértice intermediário v. Em outras palavras, cortar uma parada intermediária nunca aumenta o custo de um caminho. Formalizamos

essa noção com a definição de *desigualdade triangular*.

Desigualdade Triangular. Seja $G = (V, E)$ um grafo e uma função $c : V \times V \to \mathbb{Q}^+$ de custo entre os vértices do grafo. Dizemos que a função de custo c obedece à desigualdade triangular se, para todos os vértices $u, v, w \in V$,

$$c(u, w) \leq c(u, v) + c(v, w).$$

A desigualdade triangular é uma desigualdade natural pois em muitas aplicações é satisfeita de forma automática. Por exemplo, se os vértices de um grafo são pontos no plano e o custo de viajar entre dois vértices é a distância euclidiana comum entre eles, então a desigualdade triangular é satisfeita. Como afirmamos acima, iremos supor que a desigualdade triangular é válida para todos os problemas da dissertação.

Nesta dissertação, iremos investigar o problema *Rent-or-Buy* (RorB), assim como variantes e problemas relacionados, como o problema *Connected Facility Location* (CFL) que é um caso genérico do Rent-or-Buy com algumas modificações. O problema Rent-or-Buy é definido formalmente abaixo:

Problema 2.1 (Rent-or-Buy (RorB))**.** *Dados um grafo não orientado $G = (V, E)$, com custos $c_e \geq 0$ para cada aresta $e \in E$, um conjunto $D \subseteq V$ de* ***demandas****, cada demanda $j \in D$ com peso $d_j \geq 0$ e um parâmetro $M > 1$, encontrar um conjunto $F \subseteq V$ de facilidades a serem abertas e uma atribuição das demandas às facilidades que foram abertas. Além disso, devemos ter um grafo conexo $T \subseteq G$ que gera F (sem perda de generalidade, T é uma árvore). Se tal solução atribui a demanda j à facilidade aberta $i(j) \in F$, o custo da solução (que queremos minimizar) é definido como*

$$\sum_{j \in D} d_j \cdot c_{i(j)j} + M \sum_{e \in T} c_e,$$

onde $c_{i(j)j}$ denota o caminho mínimo entre dois vértices em G.

Iremos chamar o primeiro termo da função objetivo de *custo de conexão* (ou *custo de atribuição*), e o segundo termo de *custo de Steiner* (ou *custo da árvore de Steiner*). Iremos nos referir às arestas da árvore de Steiner como *compradas*, e às arestas no menor caminho entre uma demanda j e sua facilidade aberta mais próxima $i(j)$ como *alugadas*.

O problema Connected Facility Location é uma generalização do Rent-or-Buy, definido como segue.

Problema 2.2 (Connected Facility Location (CFL))**.** *Dados um grafo não orientado $G = (V, E)$, com custos $c_e \geq 0$ para cada aresta $e \in E$, um conjunto $D \subseteq V$ de* ***demandas****,*

*um conjunto $F \subseteq V$ de **facilidades**, cada demanda $j \in D$ com peso $d_j \geq 0$, cada facilidade i com custo de abertura f_i e um parâmetro $M > 1$, encontrar um conjunto $F' \subseteq F$ de facilidades a serem abertas e uma atribuição das demandas às facilidades que foram abertas. Além disso, devemos ter um grafo conexo $T \subseteq G$ que gera F (sem perda de generalidade, T é uma árvore). Se tal solução atribui a demanda j à facilidade aberta $i(j) \in F'$, o custo da solução (que queremos minimizar) é definido como*

$$\sum_{i \in F'} f_i + \sum_{j \in D} d_j \cdot c_{i(j)j} + M \sum_{e \in T} c_e,$$

onde $c_{i(j)j}$ denota o caminho mínimo entre dois vértices em G.

2.3 Uma motivação na prática

Informalmente falando, no Connected Facility Location, temos um conjunto de demandas e um conjunto de facilidades a serem abertas. Devemos escolher quais facilidades abrir, pagando seus preços de abertura, e então conectar as demandas às facilidades que foram abertas. Além disso, as facilidades que foram abertas devem ser conectadas por uma árvore de Steiner.

O problema Connected Facility Location surge naturalmente em diversas situações de projeto de redes. Krick et al. [21] consideram o problema de gerência de dados/caches. Neste problema, temos usuários enviando pedidos de leitura e escrita para objetos de dados. Cada objeto deve ser armazenado em um módulo de memória, pagando um certo custo de armazenamento – um objeto pode ser replicado e armazenado em múltiplas localizações. Dado uma disposição de objetos, um pedido de leitura enviado para um objeto no nodo j é servido pela localização mais próxima $i(j)$, que contém uma cópia do objeto requisitado. No entanto, um pedido de escrita precisa atualizar *todas* as cópias do objeto. Krick et al. [21] mostraram que com uma pequena perda de desempenho, isto pode ser modelado com uma única árvore de multi-difusão (*multicast*) que conecta todas as localizações que contém uma cópia do objeto. Um pedido de escrita em j primeiramente envia uma mensagem para $i(j)$ o qual então inicia a atualização de todas as cópias do objeto através da árvore de multi-difusão. O objetivo é encontrar uma disposição de objetos para módulos de memória que minimize a soma dos custos de requisição de leitura, escrita e armazenamento. Isto é exatamente a estrutura do Connected Facility Location. As facilidades são os módulos de memória e os clientes são os nodos que enviam pedidos de escrita/leitura. O custo da facilidade é o custo de armazenamento associado com o módulo de memória. A demanda de um cliente é o número de requisições feitas pelo nodo. Aqui, a exigência de conectividade é imposta pela necessidade de manter a consistência de dados.

2.4 Outros problemas

A seguir, vamos definir alguns problemas que têm ligações estreitas com os problemas Connected Facility Location e Rent-or-Buy. Esses problemas serão particularmente úteis em alguns exemplos de técnicas que utilizaremos, pois são problemas mais simples que o CFL e o RorB.

O problema *Facility Location* é muito parecido com o Connected Facility Location, mas sem a necessidade de ter que conectar as facilidades que foram abertas pelo algoritmo. Abaixo, sua definição.

Problema 2.3 (Facility Location)**.** *Dados um grafo não orientado $G = (V, E)$, com custos $c_e \geq 0$ para cada aresta $e \in E$, um conjunto $D \subseteq V$ de* ***demandas****, um conjunto $F \subseteq V$ de* ***facilidades****, cada facilidade i com custo de abertura f_i, encontrar um conjunto $F' \subseteq F$ de facilidades a serem abertas e uma atribuição das demandas às facilidades que foram abertas. Se tal solução atribui a demanda j à facilidade aberta $i(j) \in F'$, o custo da solução (que queremos minimizar) é definido como*

$$\sum_{i \in F'} f_i + \sum_{j \in D} c_{i(j)j},$$

onde $c_{i(j)j}$ denota o caminho mínimo entre dois vértices em G.

O problema da *árvore geradora mínima* é bastante conhecido na área de otimização combinatória. Apesar de ser um problema resolvido em tempo polinomial, é sempre útil como parte da solução de alguns problemas de projeto de redes NP-difíceis.

Problema 2.4 (Árvore Geradora Mínima)**.** *Dados um grafo não orientado $G = (V, E)$, com custos $c_e \geq 0$ para cada aresta $e \in E$, encontrar um subgrafo conexo $T \subseteq G$ que gera V (sem perda de generalidade, T é uma árvore), ou seja, um subgrafo que conecte todos os vértices. O custo da solução (que queremos minimizar) é definido como*

$$\sum_{e \in T} c_e,$$

ou seja, queremos minimizar o custo da árvore que gera V.

O problema da *árvore de Steiner* é parecido com o problema da árvore geradora mínima, no entanto, queremos cobrir somente um subconjunto dos vértices do grafo. Apesar de serem bem parecidos, o problema da árvore de Steiner é NP-difícil. Note que este problema é um caso particular do Connected Facility Location e do Rent-or-Buy.

Problema 2.5 (Árvore de Steiner)**.** *Dados um grafo não orientado $G = (V, E)$, com custos $c_e \geq 0$ para cada aresta $e \in E$, um conjunto $D \subseteq V$ de* ***demandas*** *(chamados*

também na literatura de vértices terminais*), encontrar um subgrafo conexo $T \subseteq G$ que cobre D (sem perda de generalidade, T é uma árvore), ou seja, um subgrafo que conecte todas as demandas. O custo da solução (que queremos minimizar) é definido como*

$$\sum_{e \in T} c_e,$$

ou seja, queremos minimizar o custo da árvore que gera D.

Iremos apresentar, a seguir, algumas notações que iremos utilizar no decorrer da dissertação. Seja S um conjunto de arestas de uma possível solução e c uma função de custos nas arestas. Então,

$$\text{val}(S) = \sum_{e \in S} c_e.$$

Usando esta notação, se T^* é o conjunto de arestas de uma solução ótima, então $\text{val}(T^*)$ é o valor da solução ótima.

Para os problemas Rent-or-Buy e Connected Facility Location, denotamos por $\text{ccon}(S)$ o custo de conexão (custo de atribuição) de uma dada solução S e por $\text{cste}(S)$ o custo de Steiner da solução S.

2.5 Introdução à técnica primal-dual

O objetivo desta seção é fazer uma breve introdução à técnica primal-dual. Em vários capítulos da dissertação, iremos descrever algoritmos baseados nesta técnica. No Capítulo 3, utilizaremos esta técnica para obter algoritmos aproximados para os problemas Rent-or-Buy (ver Problema 2.1) e Connected Facility Location (ver Problema 2.2). No Capítulo 5 usaremos esta técnica para obter um algoritmo exato para o problema da árvore geradora mínima (ver Problema 2.4). No Capítulo 6, os problemas Facility Location (ver Problema 2.3), Rent-or-Buy e Connected Facility Location serão resolvidos utilizando as idéias da técnica primal-dual sob o ponto de vista do compartilhamento de custos.

Um algoritmo primal-dual garante uma razão de aproximação que está restrita ao *gap* de integralidade de uma formulação do problema em programação linear inteira. O *gap* de integralidade de uma formulação é a razão do pior caso, sobre todas as instâncias, entre os valores de um programa linear inteiro e o valor correspondente do programa linear relaxado. Como a garantia de aproximação de um algoritmo primal-dual é obtida comparando as soluções primais e duais viáveis obtidas pelo algoritmo, sua aproximação nunca pode ser melhor que o *gap* de integralidade da formulação utilizada. Em outras palavras, se obtivermos um fator de aproximação α utilizando a técnica primal-dual, então o *gap* de integralidade não é maior que α.

Vamos considerar o seguinte programa linear primal, escrito na forma padrão. Queremos encontrar x que

$$\begin{aligned} \min & \sum_{j=1}^{n} c_j x_j \\ \text{sujeito a } & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \; i = 1, \ldots, \; m, \\ & x_j \geq 0, \qquad\qquad j = 1, \ldots, n, \end{aligned} \tag{P}$$

onde a_{ij}, b_i e c_j são valores constantes dados como entrada. O programa dual consiste em encontrar y que

$$\begin{aligned} \max & \sum_{i=1}^{m} b_i y_i \\ \text{sujeito a } & \sum_{i=1}^{m} a_{ij} y_i \leq c_j, \; j = 1, \ldots, n, \\ & y_i \geq 0, \qquad\qquad i = 1, \ldots, m. \end{aligned} \tag{D}$$

Dizemos que um vetor x é viável em (P) se x satisfaz todas as desigualdades em (P). Da mesma forma, dizemos que um vetor y é viável em (D) se y satisfaz todas as desigualdades em (D).

A *maioria* dos algoritmos de aproximação que utilizam a técnica primal-dual em sua execução garantem um conjunto de condições e relaxam apropriadamente outro conjunto de condições. Na seguinte descrição, vamos obter ambas as situações, relaxando ambas as condições. Se, eventualmente, as condições primais são satisfeitas, fazemos $\alpha = 1$, e se as condições duais são satisfeitas, fazemos $\beta = 1$.

Condições de folgas aproximadas primais:

Seja $\alpha \geq 1$. Para cada $1 \leq j \leq n$: ou $x_j = 0$ ou $c_j/\alpha \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$.

Condições de folgas aproximadas duais:

Seja $\beta \geq 1$. Para cada $1 \leq i \leq m$: ou $y_i = 0$ ou $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta \cdot b_i$.

Proposição 2.6. *Se o vetor x é viável em um programa linear primal e o vetor y é viável no dual deste programa linear, ambos satisfazendo as condições de folgas aproximadas primais e duais mostradas acima, então*

$$\sum_{j=1}^{n} c_j x_j \leq \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i y_i.$$

Demonstração.

$$\sum_{j=1}^{n} c_j x_j \leq \alpha \sum_{j=1}^{n} \Big(\sum_{i=1}^{m} a_{ij} y_i \Big) x_j = \alpha \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} a_{ij} x_j \Big) y_i \leq \alpha\beta \sum_{i=1}^{m} b_i y_i.$$

As desigualdades seguem das condições de folgas aproximadas, e a igualdade é simplesmente a troca de ordem do somatório. ■

O algoritmo primal-dual inicia com um vetor x *inviável* em (P), e um vetor y *viável* em (D). Esses vetores são geralmente os vetores triviais $x = 0$ e $y = 0$. Assim, durante sua execução, o algoritmo "melhora a viabilidade"[1] do vetor primal x e a "otimalidade"[2] do vetor dual y, garantindo que, no fim, obtenhamos o vetor x primal viável e todas as condições de folgas aproximadas mostradas acima sejam satisfeitas (com uma escolha apropriada de α e β). A viabilidade do vetor x é sempre melhorada com números inteiros, garantindo que no final o vetor obtido seja um vetor inteiro. A cada passo do algoritmo usamos os vetores primais e duais para decidir o que devemos fazer, o vetor primal em um determinado passo é usado para a melhoria do vetor dual, e vice versa. No final, o valor da função objetivo usando o vetor dual obtido é usada como limitante inferior do valor da solução ótima (no caso do dual ser de maximização e o primal ser minimização, devido ao Teorema Forte da Dualidade), e pela Proposição 2.6, a aproximação garantida pelo algoritmo é de $\alpha \cdot \beta$.

Vale observar que nem todos algoritmos aproximados que utilizam a técnica primal dual obtém sua aproximação baseada no esquema de folgas aproximadas descritas acima. Existem diversas variações de demonstrações para algoritmos primais-duais, neste documento estamos interessados somente na técnica de aproximação apresentada.

[1] A rigor, não é correto utilizar a expressão "melhora a viabilidade", pois um vetor ou é viável, ou é inviável. Ao utilizar esta expressão estamos cometendo um abuso no rigor da formalidade. O que estamos tentando dizer a expressão "melhora a viabilidade" é que a cada passo do método primal-dual, um vetor fica cada vez mais próximo de se tornar um vetor viável.

[2] Da mesma forma que a observação quanto à expressão "melhora a viabilidade", a expressão melhorar a otimalidade significa obter a cada passo um vetor mais próximo da solução ótima.

2.6 Facility Location

Nesta seção, iremos comentar sobre o problema Facility Location, apresentando uma formulação em programação linear e um algoritmo baseado na técnica primal-dual. Além disso, vamos enunciar alguns resultados decorrentes do uso do algoritmo apresentado. O algoritmo e os resultados desta seção serão utilizados no Capítulo 6 como exemplo. Assim sendo, não há necessidade de aprofundar-nos nas demonstrações desses resultados, e estes estarão omitidos neste documento.

A seguir apresentamos uma formulação do problema Facility Location (problema 2.3). Seja F o conjunto de facilidades, D o conjunto de demandas, cada conexão ij, onde $i \in F$ e $j \in D$ com custo c_{ij} e custos para abrir uma facilidade $i \in F$ com custo f_i. Com estes dados em mãos, o problema Facility Location consiste em encontrar x e y que:

$$\begin{array}{lll} \min & \sum_{i\in F}\sum_{j\in D} c_{ij}x_{ij} + \sum_{i\in F} y_i f_i & \\ \text{s.a.} & \sum_{i\in F} x_{ij} = 1 & \forall j \in D, \\ & x_{ij} \leq y_i & \forall i \in F,\ j \in D, \\ & x_{ij} \in \{0,1\} & \forall i \in F,\ j \in D, \\ & y_i \in \{0,1\} & \forall i \in F. \end{array} \tag{2.1}$$

2.6.1 Algoritmo de Jain e Vazirani

O algoritmo que iremos apresentar foi proposto por Jain e Vazirani em [18]. Ele consiste em uma 3-aproximação para o problema Facility Location utilizando a técnica primal-dual.

Considere a relaxação do programa linear inteiro (2.1) e o dual respectivo:

$$\begin{array}{lll} \min & \sum_{i\in F}\sum_{j\in D} c_{ij}x_{ij} + \sum_{i\in F} y_i f_i & \\ \text{s.a.} & \sum_{i\in F} x_{ij} \geq 1 & \forall j \in D, \\ & x_{ij} \leq y_i & \forall i \in F,\ j \in D, \\ & 0 \leq x_{ij} \leq 1 & \forall i \in F,\ j \in D, \\ & 0 \leq y_i \leq 1 & \forall i \in F, \end{array} \tag{2.2}$$

e

$$\begin{array}{lll} \max & \sum_{j\in D} \alpha_j & \\ \text{s.a.} & \alpha_j - \beta_{ij} \leq c_{ij} & \forall i \in F,\ j \in D, \\ & \sum_{j\in D} \beta_{ij} \leq f_i & \forall i \in F, \\ & \alpha_j \geq 0 & \forall j \in D, \\ & \beta_{ij} \geq 0 & \forall i \in F,\ j \in D. \end{array} \tag{2.3}$$

O algoritmo começa os vetores (α, β) viáveis para o programa dual e uma série de variáveis que representam soluções (x, y), a princípio inviáveis, do programa primal. O

algoritmo é composto de duas fases e, em cada iteração da primeira fase, os valores de α e β aumentam, mantendo viabilidade, enquanto (x, y) não forem vetores viáveis e inteiros para (2.2). Na segunda fase, o algoritmo faz uma espécie de aglomeração nas facilidades gerando os vetores (x, y) inteiros do programa primal (2.2) e vetores (α, β) viáveis para o programa dual (2.3), respeitando a equação

$$\sum c_{ij}x_{ij} + 3\sum y_i f_i \leq 3\sum \alpha_j, \tag{2.4}$$

de onde se pode concluir que o algoritmo é uma 3-aproximação para o problema Facility Location.

No algoritmo que apresentaremos abaixo, consideramos que o grafo é bipartido, com uma partição para os clientes e outra para as facilidades. O custo das arestas que ligam as partições são obtidos calculando no grafo original os custos dos caminhos mínimos entre os clientes e facilidades.

A seguir apresentamos uma descrição dos passos de cada fase.

Fase 1

Para cada cliente j, mantenha sua variável dual α_j crescendo até que ela esteja conectada temporariamente a uma facilidade.

É definida, nesta fase, uma noção de tempo t. Se um evento 1 ocorre antes de um evento 2, $t_1 \leq t_2$. A fase inicia com tempo igual a zero. Cada cliente é definido como não conectado. O algoritmo cresce a variável dual α_j para cada cliente não conectado j, crescendo de 1 a cada unidade de tempo. Quando temos $\alpha_j = c_{ij}$ para alguma aresta (i, j), esta aresta é declarada justa. Neste ponto β_{ij} será chamado *especial* e passa a crescer uniformemente em conjunto com α_j até que uma das restrições de β_{ij} em (2.3) se torne justa. Quando temos $\sum_j \beta_{ij} = f_i$, o algoritmo declara a facilidade i temporariamente aberta. Além disso, todos os clientes com arestas justas a i são declarados conectados e a facilidade i passa a ser chamada de testemunha de conexão destes clientes. No futuro, assim que um cliente tiver uma aresta justa a i, será automaticamente conectado a ela, e i será sua testemunha de conexão.

Se vários eventos ocorrerem ao mesmo tempo, o algoritmo escolhe apenas um dos eventos e continua a execução. A primeira fase termina quando todos os clientes estiverem conectados.

Fase 2

Seja F_t o conjunto de facilidades temporariamente abertas na fase 1 e T o subgrafo de G gerado pelas arestas que estavam justas no final da fase 1. Seja T^2 o grafo obtido a partir de T, com o mesmo conjunto de vértices e uma aresta (u, v) pertence a T^2 se e somente se existe um caminho de comprimento 2 em T entre u e v. Seja H um subgrafo de T^2 induzido em F_t. As facilidades em F_t serão consideradas na ordem em que foram

declaradas temporariamente abertas na fase 1 e um subconjunto maximal independente, I, será obtido de H da seguinte forma. Enquanto estiver considerando a facilidade i, se nenhum vizinho de i em H foi escolhido, i é escolhido. Caso contrário, o vizinho que pertence a I é declarado testemunha de i e i não é aberto. Todas as facilidades em I são declaradas abertas. A função $\phi : D \rightarrow F$ é obtida da seguinte forma: para cada $j \in D$, se existe uma aresta (i, j) justa para algum $i \in I$, $\phi(j) = i$, caso contrário $\phi(j) =$ testemunha(i).

Finalmente, $x_{ij} = 1$ se e somente se $\phi(j) = i$ e $y_i = 1$ se e somente se $i \in I$.

É possivel mostrar que o algoritmo se afasta do ótimo quando os clientes são conectados indiretamente, via testemunha. Neste caso temos, através da desigualdade triangular, que $c_{ij} \leq c_{ij'} + c_{i'j'} + c_{i'j}$, para todo i' e j'.

Deve existir i' e j' tal que testemunha$(i) = i'$ e i' é a testemunha de conexão de j'. Pela forma como o algoritmo opera, veja mais detalhes em [18], temos que $c_{ij'} + c_{i'j'} + c_{i'j} \leq 3\alpha_j$ o que implicará na validação da equação (2.4).

Algoritmo 2.1: Algoritmo-JV

Entrada: grafo $G = (V, E)$, facilidades F, demandas D
Saída: facilidades abertas I, atribuições ϕ
início
  {**Inicializações**};
  $A \leftarrow D$; `/* conjunto de demandas ativas */`
  $S \leftarrow \emptyset$; `/* conjunto de arestas especiais */`
  $R \leftarrow \emptyset$; `/* conjunto de arestas especiais inativas */`
  $F_t \leftarrow \emptyset$; `/* facilidades temporariamente abertas */`
  para cada $j \in D$ **faça** $\alpha_j = 0$;
  para cada $j \in D$ *e* $i \in F$ **faça** $\beta_{ij} = 0$;
  {**Fase 1**};
  enquanto $A \neq \emptyset$ **faça**
    aumente uniformemente α_j e β_{ij} para $j \in A$ e $ij \in S \setminus R$ até que
    (a) $\alpha_j = c_{ij}$ para algum $ij \in (F_t \times A) \setminus S$, ou
    (b) $\alpha_j = c_{ij}$ para algum $ij \in (F \setminus F_t \times A) \setminus S$, ou
    (c) $\sum_{j \in D} \beta_{ij} = f_i$ para algum $i \in F \setminus F_t$.
    se *(a) foi satisfeita para algum* ij **então**
      $\tau(j) \leftarrow i$; `/* testemunha de` j `*/`
      $A \leftarrow A \setminus \{j\}$;
    senão se *(b) foi satisfeita para algum* ij **então**
      $S \leftarrow S \cup \{ij\}$;
    senão se *(c) foi satisfeita para algum* i **então**
      $A \leftarrow A \setminus \{j : ij \in S\}$;
      $R \leftarrow R \cup \{ij : ij \in S\}$;
      $F_t \leftarrow F_t \cup \{i\}$;
      para cada j *tal que* $ij \in S$ **faça** $\tau(j) = i$;
  {**Fase 2**};
  $T \leftarrow G[S]$;
  $H = (V_H, E_H)$ onde $V_H = F_t$ e $E_H = \{i'i'' : \exists j \in D, i'j \in S \text{ e } i''j \in S\}$;
  $I \leftarrow$ (conjunto independente maximal de H);
  para todo $j \in D$ **faça**
    se $\tau(j) \in I$ **então**
      $\phi(j) \leftarrow \tau(j)$; `/*` $(\tau(j), j)$ `é conexão direta */`
    senão
      Seja $i \in I$ tal que $(i, \tau(j)) \in H$;
      $\phi(j) \leftarrow i$; `/*` ij `é conexão indireta */`
  devolva I, ϕ
fim

O tempo de execução deste algoritmo é $O(m \log m)$, que é o tempo de se ordenar as arestas segundo seu custo.

Teorema 2.7. *O Algoritmo 2.1 é um algoritmo* 3*-aproximado para o problema Facility Location.*

2.7 Árvore de Steiner

Nesta seção, iremos comentar sobre o problema da árvore de Steiner, apresentando uma formulação em programação linear, um algoritmo baseado na técnica primal-dual e um algoritmo combinatório. Além disso, vamos enunciar alguns resultados decorrentes do uso destes algoritmos. Os algoritmos e os resultados desta seção serão utilizados em vários capítulos no decorrer da dissertação. Assim sendo, não há necessidade de aprofundar-nos nas demonstrações desses resultados, e estes estarão omitidos neste documento.

A seguir apresentamos uma formulação do problema da árvore de Steiner (Problema 2.5). Seja $D \subseteq V$ o conjunto de vértices terminais, cada aresta e com custo c_e. Dizemos que um conjunto $S \subset V$ é *válido* se $S \cap D \neq D$ e $S \cap D \neq \emptyset$. Seja $\delta(S)$ as arestas pertencentes ao corte de S, ou seja, $\delta(S) = \{(u, v) \in E | u \in S, v \in V \setminus S\}$. Com estes dados em mãos, o programa linear inteiro para o problema da árvore de Steiner mínima consiste em encontrar x que

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e & \text{(PI)} \\ \text{s.a.} \quad & \sum_{e : e \in \delta(S)} x_e \geq 1, \qquad \forall S \text{ válido}, \\ & x_e \in \{0, 1\}, \qquad \forall e \in E. \end{aligned}$$

Dizemos que um dado conjunto $S \subset V$ é um *conjunto violado* se não respeita a restrição do programa linear inteiro de haver pelo menos uma aresta em $\delta(S)$. No algoritmo que iremos descrever a seguir, é adicionada uma aresta a cada passo (explicaremos melhor depois), fazendo com que alguns conjuntos deixem de ser violados. Dizemos que um dado conjunto $S \subset V$ é um *conjunto minimal violado* (CMV) se S é violado e não existe nenhum outro conjunto $S' \subset S$ que é violado. A Figura 2.1 ilustra essas idéias.

2.7.1 Algoritmo de Goemans e Williamson

O algoritmo que iremos apresentar foi proposto por Goemans e Williamson em [9]. Ele consiste em uma 2-aproximação para o problema da árvore de Steiner utilizando a técnica

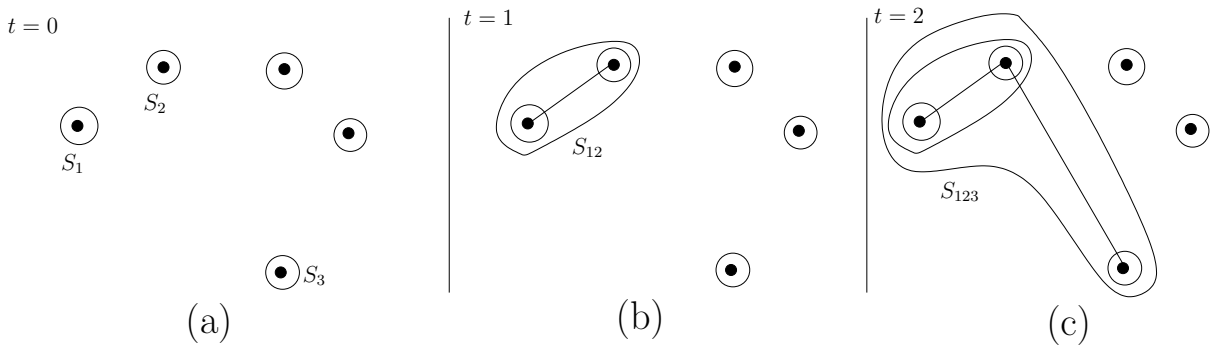

Figura 2.1: (a)Inicialmente, os conjuntos minimais violados são os conjuntos unitários, pois não há aresta no corte dos CMV. Além disso, qualquer subconjunto de dois ou mais vértices forma um conjunto violado, mas não um CMV. (b) Uma aresta é adicionada e os conjuntos S_1 e S_2 passam a ser não violados e um outro conjunto minimal violado, que chamamos de S_{12}, contendo S_1 e S_2 é criado. (c) Uma aresta é adicionada e o conjunto S_{12} torna-se não violado. Um outro conjunto minimal violado, que chamamos de S_{123}, contendo S_{12} e S_3 é criado.

primal-dual.

A seguir, temos a relaxação do programa linear inteiro (PI) e o dual respectivo:
Primal:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{s.a.} \quad & \sum_{e : e \in \delta(S)} x_e \geq 1, \qquad \forall S \text{ válido}, \\ & x_e \geq 0, \qquad \forall e \in E, \end{aligned}$$

Dual:

$$\begin{aligned} \max \quad & \sum_{S \text{ válido}} y_S \\ \text{s.a.} \quad & \sum_{S : e \in \delta(S)} y_S \leq c_e, \qquad \forall e \in E \\ & y_S \geq 0, \qquad \forall S \text{ válido}. \end{aligned}$$

Dizemos que a aresta e *sente* o dual y_S se $y_S > 0$ e $e \in \delta(S)$. Dizemos que a aresta e está *justa* se a quantidade total do dual que ela sente é igual ao seu custo. O programa linear dual está tentando maximizar a soma das variáveis duais y_S sujeito à condição de que nenhuma aresta sente mais dual que o seu custo, i.e., nenhuma aresta está justa demais.

Abaixo, iremos mostrar o algoritmo de Goemans e Williamson, que é baseado na técnica primal-dual. Começando com as soluções triviais primal e dual o algoritmo iterativamente "melhora a viabilidade" da solução primal e a "otimalidade" da solução dual. Quando uma aresta fica justa, ela é incluída no conjunto T. É facil ver que os CMV devem ser disjuntos. De fato, quando uma aresta que liga dois CMV's se torna justa, então os dois conjuntos se tornam não-violados e um novo CMV que "engloba" os dois anteriores é formado.

Em cada unidade de tempo, o algorimo cresce as variáveis duais de forma unitária. Em uma dada iteração, o algoritmo encontra todos os CMV's e aumenta suas variáveis duais até que uma nova aresta, digamos e, fique justa. Agora, a iteração atual termina e a aresta e é adicionada à lista T. O algoritmo continua até que não exista mais nenhum CMV. As arestas em F formam uma solução viável para o problema da árvore de Steiner.

Algoritmo 2.2: Algoritmo-GW

Entrada: grafo $G = (V, E)$, demandas $D \subseteq V$;
Saída: árvore de Steiner T;
início
  $T \leftarrow \emptyset$;
  para cada $S \subseteq V$ **faça**
    $y_S \leftarrow 0$
  enquanto $\exists$ *CMV's* **faça**
    Encontre todos os CMV's;
    Aumente uniformemente as variáveis duais y_S, para todos CMV's, até alguma aresta e ficar justa;
    $T \leftarrow T \cup \{e\}$;
  devolva T
fim

Teorema 2.8. *O Algoritmo 2.2 é um algoritmo 2-aproximado para o problema da árvore de Steiner.*

Observação 2.9. Em alguns casos, durante a dissertação, iremos nos referir ao algoritmo primal-dual para a *árvore de Steiner enraizada*. A idéia é a mesmo do algoritmo descrito acima, porém dizemos que um conjunto $S \subset V$ é *válido* se $r \notin S$ e $S \cap D \neq \emptyset$, onde $r \in D$ é um vértice terminal que é escolhido como raiz da árvore.

2.7.2 Heurística da árvore geradora mínima

Um outro resultado de aproximação que iremos utilizar mais a frente na dissertação é um algoritmo combinatório para a árvore de Steiner. Este resultado foi obtido independente-

mente por Choukhmane [5], Iwainsky, Canuto, Taraszow e Villa [16], Kou, Markowsky e Berman [20] e Plesník [29].

O algoritmo calcula uma árvore geradora mínima em um grafo completo $G' = (V', E')$ que é criado da seguinte maneira. O conjunto de vértices $V' = D$, onde D são os vértices terminais do grafo original G e os custos $c_{(u,v)}$ para $(u, v) \in E'$ são calculadas de acordo com o caminho mínimo entre u e v no grafo original G. Em outras palavras, o grafo G' é obtido pelo fechamento de G, tal que as arestas em G' correspondem aos caminhos mínimos de G. O algoritmo completo é descrito abaixo.

Algoritmo 2.3: Algoritmo-AGM

Entrada: grafo $G = (V, E)$, demandas $D \subseteq V$;
Saída: árvore de Steiner T;
início
  obtenha $G' = (V', E')$ da seguinte maneira;
    $V' \leftarrow D$;
    $E' \leftarrow \{(u, v) | u \in D, v \in D\}$;
  para todo $(u, v) \in E'$ **faça**
    $c_{(u,v)} \leftarrow$ custo do caminho mínimo entre u e v em G;
  seja T uma árvore geradora mínima de G';
  mapeie a árvore T novamente para G, substituindo arestas por caminhos;
  retire de T as arestas supérfluas para a solução, como arestas múltiplas e ciclos;
  devolva T
fim

Teorema 2.10. *O Algoritmo 2.3 é um algoritmo 2-aproximado para o problema da árvore de Steiner.*

Capítulo 3

Técnica primal-dual

Neste capítulo, iremos utilizar a técnica primal-dual para obter algoritmos de aproximação e está baseado nos resultados obtidos em [34]. Trata-se de um algoritmo de aproximação para o problema Rent-or-Buy, obtendo um fator de aproximação de 4.55, que é atualmente o melhor resultado de aproximação utilizando algoritmos determinísticos para este problema. Um resultado melhor foi obtido utilizando algoritmos probabilísticos, e será descrito no Capítulo 4. Além disso, ainda em [34], foi proposto um algoritmo de aproximação para o Connected Facility Location, obtendo um fator de aproximação 8.55, que é atualmente o melhor resultado para o problema. Iremos apresentar esse algoritmo para o CFL neste capítulo.

3.1 Rent-or-Buy

Baseado nas definições dos problemas, apresentadas no Capítulo 2, vamos fazer as seguintes suposições.

Suposição 3.1 (**demandas unitárias**)**.** Por questões de simplicidade, iremos assumir que todas as demandas $d_j = 1$.

Na Subseção 3.2.4 é mostrado como fazer para abandonar esta suposição.

Suposição 3.2 (**raiz na solução**)**.** Iremos assumir que sabemos de antemão que uma facilidade r (chamada de raiz) está aberta e pertence à árvore de Steiner construída na solução ótima. O caso geral pode ser obtido simplesmente testando todas as $|V|$ possibilidades para r.

Primeiramente, vamos considerar o *problema Rent-or-Buy*. Na Seção 3.2 mostraremos um algoritmo para o Connected Facility Location, cuja idéia é parecida com a idéia que iremos apresentar para o Rent-or-Buy.

Seja $S \subseteq V$. Denotamos por $\delta(S)$ as arestas que têm um extremo em S e outro extremo em $V \setminus S$. Assim, para o Rent-or-Buy, temos a seguinte formulação em programação linear inteira, onde queremos encontrar os vetores x e y que

$$\min \sum_{j \in D} \sum_{i \in F} c_{ij} x_{ij} + M \sum_{e \in E} c_e z_e \tag{P1}$$

$$\text{s.a.} \sum_{i \in F} x_{ij} \geq 1 \qquad \forall j \in D, \tag{3.1}$$

$$\sum_{i \in S} x_{ij} \leq \sum_{e \in \delta(S)} z_e \qquad \forall S \subseteq V, r \notin S, \forall j \in D, \tag{3.2}$$

$$x_{ij}, z_e \in \{0, 1\} \qquad \forall j \in D, \forall i \in F, \forall e \in E.$$

Lembrando que o conjunto $F = V$ no problema Rent-or-Buy, ou seja, todos os vértices do grafo são possíveis facilidades, e além disso, não há custos para abrir facilidades. Antes de apresentar a formulação dual, vamos explicar brevemente o significado da formulação anterior. As variáveis x_{ij} dizem respeito à atribuição de demandas às facilidades, ou seja, $x_{ij} = 1$ se a demanda j está atribuída à facilidade i, $x_{ij} = 0$ caso contrário. As variáveis z_e estão relacionadas com a construção da árvore de Steiner que conecta as facilidades escolhidas, ou seja, $z_e = 1$ se a aresta e pertence à árvore de Steiner, $z_e = 0$ caso contrário. Portanto, a função objetivo é a somatória dos custos de conexão com os custos da árvore de Steiner (multiplicados pela constante M). Lembrando, como explicado no Capítulo 2, o custo de conexão é o custo relacionado às demandas que se conectaram a alguma facilidade e o custo de Steiner é o custo relacionado à árvore de Steiner. As restrições (3.1) dizem que, para cada demanda j, devemos ter pelo menos uma facilidade a qual j é atribuída. Como o problema é de minimização, uma demanda nunca estará associada a duas ou mais facilidades, pois de acordo com as restrições (3.1) seria desnecessário e só acarretaria no aumento do valor da função objetivo. Portanto, em qualquer solução para este programa linear, uma demanda está atribuída a exatamente uma facilidade. Para as restrições (3.2), relacionadas à construção da árvore de Steiner, fixamos uma demanda j e um conjunto de vértices S, onde a raiz $r \notin S$. Note que, como argumentado acima, o somatório do lado esquerdo é no máximo um, podendo ser zero caso o conjunto S fixado não inclua a facilidade a qual j foi atribuído. Temos então dois casos:

(a) No caso em que a somatória do lado esquerdo é igual a zero, então a restrição é sempre satisfeita, pois o somatório do lado direito nunca é negativo.

(b) No caso em que a somatória do lado esquerdo é igual a um, o lado direito deve ser

pelo menos um, pois deve existir um caminho entre i e a raiz r (que não pertence a S).

Note que a restrição (3.2) considera todos os subconjuntos S possíveis, garantindo assim a construção da árvore de Steiner de custo mínimo que conecta as facilidades.

Vamos utilizar a formulação relaxada do programa linear acima, modificando as restrições de integralidade para $x_{ij}, z_e \geq 0$.

O dual da relaxação do programa linear (P1) consiste em encontrar α e θ tais que

$$\max \sum_{j \in D} \alpha_j \tag{D1}$$

$$\text{s.a.} \quad \alpha_j \leq c_{ij} + \sum_{S \subseteq V : i \in S, r \notin S} \theta_{S,j} \quad \forall i \in F, \forall j \in D, i \neq r, \tag{3.3}$$

$$\alpha_j \leq c_{rj} \quad \forall j \in D, \tag{3.4}$$

$$\sum_{j \in D} \sum_{S \subseteq V : e \in \delta(S), r \notin S} \theta_{S,j} \leq M \cdot c_e \quad \forall e \in E, \tag{3.5}$$

$$\alpha_j, \theta_{S,j} \geq 0.$$

Para o entendimento do texto que segue, é importante ressaltar como as variáveis duais são interpretadas. Intuitivamente, α_j é a quantidade que j está pagando para fazer com que a solução do problema dual seja primal viável. A restrição (3.3) significa que parte do pagamento de α_j é para atribuir a demanda j à facilidade i, e parte é para a construção da árvore de Steiner que liga i à r.

Antes de descrever o algoritmo, iremos fazer uma *suposição* que será conveniente para analisar o algoritmo. Veremos na Subseção 3.1.4 que esta suposição pode ser abandonada, sem afetar o fator de aproximação obtido.

Suposição 3.3 (**métrica infinitesimal nas arestas**)**.** Iremos assumir que uma facilidade pode ser aberta em *qualquer lugar ao longo de uma aresta*. A métrica c é extendida para uma métrica nas localidades, que considera uma aresta $e = (u, w)$ de comprimento c_e sendo composta por infinitas arestas de tamanho infinitesimal. Assim, para pontos p na aresta e, a distância (custo) c_{up} varia continuamente e monotonicamente de 0 a c_e quando vamos de u a w, e $c_{wp} = c_e - c_{up}$. Para qualquer outro vértice $t \neq u, r \neq w$, o valor de c_{tp} é igual a $\min(c_{tu} + c_{up},\ c_{tw} + c_{wp})$. Finalmente, para quaisquer dois pontos p, q pertencentes respectivamente às arestas $e_1 = (u, w)$ e e_2, a distância c_{pq} é dada por $\min(c_{up} + c_{uq},\ c_{wp} + c_{wq})$.

A Figura 3.1 exemplifica o uso da Suposição 3.3 (métrica infinitesimal nas arestas). Como iremos ver com mais detalhes a seguir, o algoritmo cresce raios em volta das demandas, e quando M raios se interceptam num ponto, esse ponto é uma facilidade em potencial.

Os vértices em V e os pontos internos de uma aresta serão chamados de *localidades*. Usaremos o termo *facilidade* para os vértices que estão em F.

3.1.1 Visão geral do algoritmo

A seguir, explicamos a idéia do algoritmo. Suponha que cada cliente tem M unidades de demanda. Então, a função objetivo seria pelo menos

$$\min \sum_j M \sum_i c_{ij} x_{ij} + M \sum_e c_e z_e.$$

Podemos manipular o somatório, obtendo como função objetivo $\min M \sum_j \sum_i c_{ij} x_{ij} + M \sum_e c_e z_e$. Como M é uma constante multiplicativa da função objetivo, esta não afetaria uma solução final do programa linear, reduzindo a função objetivo a

$$\min \sum_{i,j} c_{ij} x_{ij} + \sum_e c_e z_e.$$

Neste caso, para todos os clientes, é possível mostrar que uma solução ótima irá abrir uma facilidade no próprio cliente e então conectá-los utilizando uma árvore de Steiner. Usando este fato, como estaremos considerando a Suposição 3.1 de demandas unitárias, então a idéia principal do algoritmo é agrupar as demandas em *aglomerados* de pelo menos M demandas, e então construir a árvore de Steiner que conecta estes aglomerados.

Inicialmente, todas as variáveis duais têm seu valor igual a 0. O algoritmo é composto de duas fases. Na primeira fase, nós agrupamos as demandas em aglomerados de tamanho M. Feito isso, passamos para a segunda fase, onde a árvore de Steiner é construída.

3.1.2 Descrição do algoritmo

Fase 1

Nesta fase, crescemos o valor das variáveis duais α_j para todas as demandas simultaneamente. Introduzimos a noção de tempo, t. No início do algoritmo, $t = 0$. A cada intervalo, o tempo é acrescido de uma unidade. A medida que o tempo cresce, também cresce o valor das variáveis duais α_j de forma uniforme. Algumas localidades podem estar no estado *possivelmente abertas*. Uma localidade se torna possivelmente aberta quando

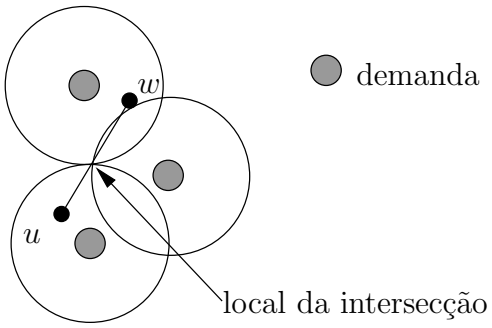

Figura 3.1: Supondo $M = 3$, quando três raios se interceptam no meio de uma aresta, uma localidade pode ser aberta neste lugar, assim como foi dito na Suposição 3.3 (métrica infinitesimal nas arestas).

M ou mais raios se interceptam nesta localidade, como veremos a seguir. Além disso, posteriormente, um subconjunto das localidades possivelmente abertas será escolhido para serem abertas na solução final do algoritmo. Quando $t = 0$, r está possivelmente aberta e todas as outras localidades estão fechadas.

A função objetivo do dual do programa linear quer aumentar o valor das variáveis duais α_j. No entanto, estas não podem crescer para sempre devido às restrições do programa linear (lembrando que queremos uma solução dual viável). Assim crescemos α_j até esta ficar *justa* em alguma localidade i. Ou seja, uma variável α_j está *justa* com relação a alguma localidade i se $\alpha_j \geq c_{ij}$. Seja S_j o conjunto de vértices aos quais j está justo em um dado instante de tempo. Ao crescer α_j, também crescemos $\theta_{S_j,j}$ na mesma razão. Isto irá garantir a viabilidade das restrições (3.3), como veremos no Lema 3.4.

As demandas podem estar em dois estados: *congeladas* e *não-congeladas*. Quando uma demanda j fica congelada, paramos de crescer sua variável dual α_j. Se a demanda j está não-congelada no instante de tempo t, então $\alpha_j = t$. Após j ser congelada, ela não mais fica justa com nenhuma outra nova localidade, i.e., uma localidade que não pertence a S_j, pois o raio de j parou de crescer. No início do algoritmo, todas as demandas estão não-congeladas.

Começamos crescendo à razão unitária a variável α_j de todas as demandas simultaneamente, até um desses eventos ocorrer (se vários eventos ocorrerem ao mesmo tempo, considere eles em qualquer ordem):

1. j **se ajusta com uma localidade** i **possivelmente aberta**: j se torna congelada.

2. **Uma localidade fechada** i **fica justa com pelo menos** M **demandas**: i fica possivelmente aberta. Todos os pontos de demandas que estão justos com i tornam-se congelados.

A cada iteração, crescemos a variável α_j somente das demandas não-congeladas. Continuamos este processo até que todas as demandas fiquem congeladas.

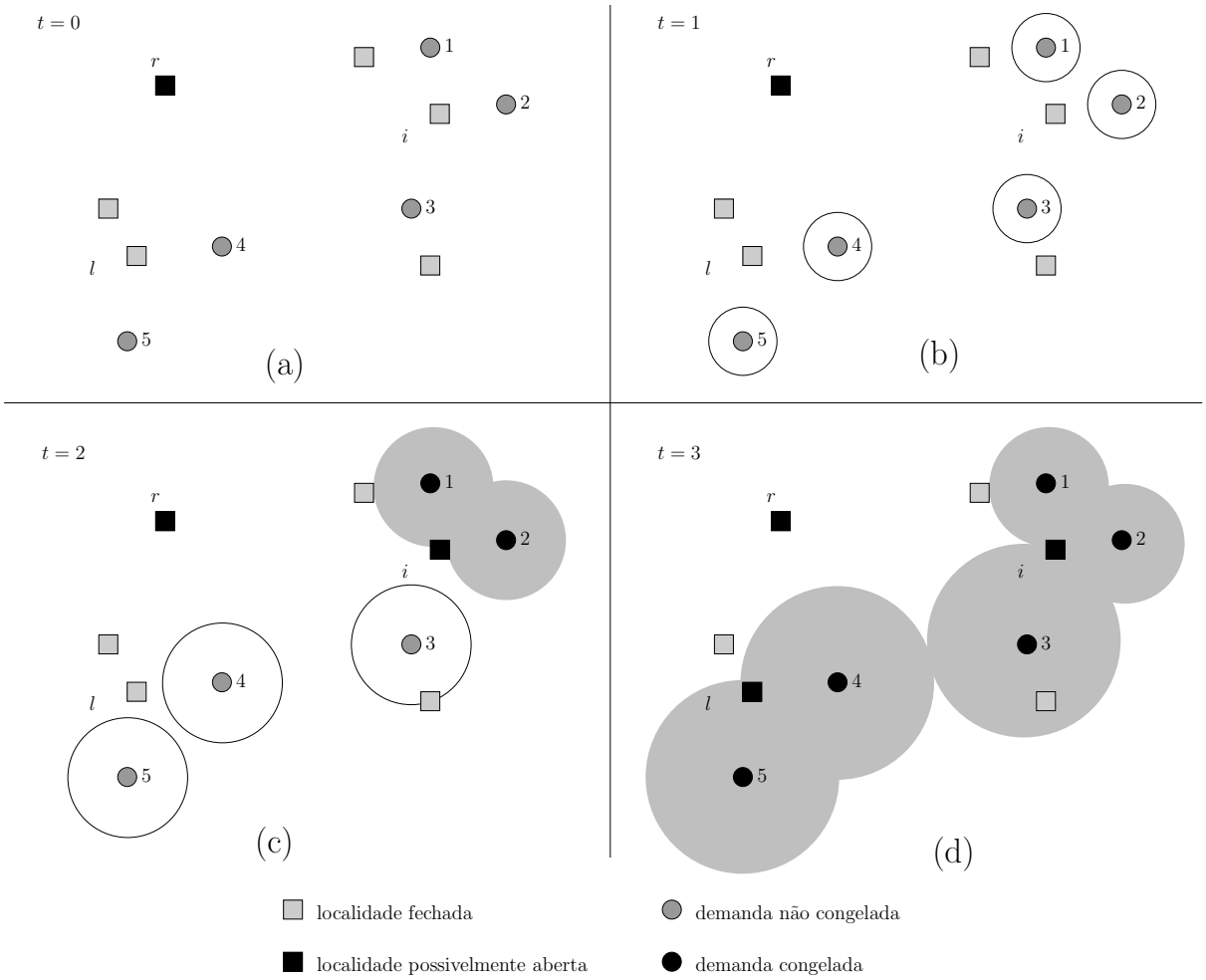

Figura 3.2: Um exemplo de execução com $M = 2$. (a) O estado inicial, (b) t=1, (c) i fica justo com as demandas 1 e 2; i fica possivelmente aberto e 1 e 2 ficam congelados, (d) A solução final. A demanda 3 alcança i e fica congelada; l fica justa com as demandas 4 e 5 e fica possivelmente aberta, fazendo com que 4 e 5 congelem.

Note que, ainda que haja um *continuum* de pontos ao longo de uma aresta, para implementar o processo descrito acima precisamos somente saber o instante em que o próximo evento irá acontecer. Isto pode ser feito monitorando, para cada aresta e cada demanda j, a parte da aresta que está justa com j.

Agora vamos decidir quais localidades devem ser abertas. Seja L o conjunto das localidades possivelmente abertas. Dizemos que as localidades $i, i' \in L$ são *dependentes* se há uma demanda j a qual está justa com ambas as localidades. Dizemos que um

conjunto de localidades é *independente* se não há duas localidades dependentes neste conjunto. Devemos então encontrar um conjunto independente maximal L' de localidades em L da seguinte forma: arranje as localidades em L em ordem do instante de tempo em que foram marcadas como possivelmente abertas. Considere as localidades nesta ordem e adicione uma localidade em L' se esta não for dependente com alguma localidade já presente em L'. Finalmente, as localidades em L' são abertas. Observe que $r \in L'$.

Associamos uma demanda j a uma localidade aberta da seguinte forma. Se j está justa com alguma localidade $i \in L'$, atribua j à i. Caso contrário, seja i a localidade em $L \setminus L'$ a qual fez com que j congelasse. Então j está justo com i. Deve haver uma localidade $i' \in L'$ aberta anteriormente tal que i e i' são dependentes. Assim, atribuímos j à i'. Denotamos por $\sigma(j)$ a localidade a qual j está atribuída.

Temos agora que construir a árvore de Steiner em L'. Antes disso, devemos aumentar o grafo G para incluir arestas incidentes às localidades abertas que não são vértices (i.e., localidades que foram abertas em algum ponto ao longo de uma aresta). Seja $\{i_1, \ldots, i_k\}$ as localidades abertas na aresta $e = (u, w)$, ordenadas pela distância com relação à u, com $i_1 \neq u$ e $i_k \neq w$. Adicionamos então as arestas $(u, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, w)$ em G.

Fase 2

Para uma localidade $i \in L'$, seja D_i o conjunto de demandas justas com i. Seja $D' = \bigcup_{i \in L' - \{r\}} D_i$. Inicialmente, fazemos $\alpha_j = 0$ (os α_j da Fase 2 assumirão um valor totalmente diferente do valor da Fase 1), para todo $j \in D$. Crescemos somente o valor α_j das demandas em D', e simulamos o algoritmo primal-dual para o problema da árvore de Steiner (enraizada) que foi apresentado no Algoritmo 2.2 do Capítulo 2 e será descrito rapidamente abaixo.

Os conjuntos minimais violados (CMV) são os conjuntos unitários $\{i\}$ para $i \in L' - \{r\}$. Para um conjunto S, defina $D_S = \bigcup_{i \in S \cap L'} D_i$. A árvore T que iremos construir começa vazia. Para cada CMV S, $j \in D_S$, a variável α_j cresce à razão de $1/|D_S|$. A variável $\theta_{S,j}$ também cresce à mesma razão. Assim, garantimos que $\sum_j \theta_{S,j}$ cresce à razão de 1 para qualquer CMV S. Note que *não* estamos assegurando a viabilidade das restrições (3.3) e (3.4), pois α_j pode ser maior que o lado direito destas restrições.

Dizemos que uma aresta *se ajusta* (ou *fica justa*) se a restrição (3.5) é satisfeita com igualdade para aquela aresta. Nós crescemos as variáveis duais até uma aresta e se ajustar. Adicionamos e em T e atualizamos os conjuntos minimais violados. Este processo continua até não haver mais conjuntos violados, i.e., continua até termos somente um componente (r está neste componente). Agora, consideramos as arestas de T na ordem inversa em que foram inseridas e removemos qualquer aresta redundante. Esta é nossa solução final.

A forma algorítmica é apresentada a seguir. Em cada iteração, seja $\mathcal{C}$ o conjunto das demandas não-congeladas.

Algoritmo 3.1: RorB-primal-dual

Entrada: grafo $G = (V, E)$, demandas $D \subseteq V$, parâmetro $M > 1$
Saída: árvore de Steiner T, facilidades L' e atribuições $\sigma(j)$
início
  {**Fase 1**};
  (Inicialização) $L \leftarrow \{r\}, \mathcal{C} \leftarrow D$;
  enquanto $\mathcal{C} \neq \emptyset$ **faça**
    aumente o tempo t até que alguma das condições abaixo sejam satisfeitas;
    se $j \in \mathcal{C}$ ***e*** $\alpha_j \geq c_{ij}$ **então**
      $D_i \leftarrow D_i \cup \{j\}$;
      se $i \in L$ **então**
        $\mathcal{C} \leftarrow \mathcal{C} - \{j\}$;
    se $i \notin L$ ***e*** $|D_i| \geq M$ **então**
      $\mathcal{C} \leftarrow \mathcal{C} - D_i$;
      $L \leftarrow L \cup \{i\}$;
    se $j \in \mathcal{C}$ **então** aumente α_j e $\theta_{S_j,j}$
  crie L' removendo as localidades dependentes em L;
  para todo $i \in L \setminus L'$ **faça**
    para todo $j \in D_i$ **faça**
      $\sigma(j) \leftarrow \{i'\}$;
  {**Fase 2**};
  (Inicialização) $T \leftarrow \emptyset$; $D' \leftarrow \bigcup_{i \in L' - \{r\}} D_i$; $\alpha_j \leftarrow 0$; $\theta_{S,j} \leftarrow 0$;
  Seja $\mathcal{S}$ a coleção dos conjuntos minimais violados;
  enquanto $\mathcal{S} \neq \emptyset$ **faça**
    se $\sum_{j \in D} \sum_{S \subseteq V : e \in \delta(S), r \notin S} \theta_{S,j} = M \cdot c_e$ **então**
      $T \leftarrow T \cup \{e\}$;
      atualiza $\mathcal{S}$;
    se $j \in D'$ aumente uma unidade em α_j e $1/|D_S|$ em $\theta_{S,j}$;
  remova todas arestas redundantes em T;
  devolva T, L' *e as atribuições* $\sigma(j)$;
fim

3.1.3 Análise

Sejam $(\alpha^{(1)}, \theta^{(1)}), (\alpha^{(2)}, \theta^{(2)})$ os valores das variáveis duais no fim das Fases 1 e 2, respectivamente.

Lema 3.4. *A solução dual* $(\alpha^{(1)}, \theta^{(1)})$ *é viável.*

Demonstração. É simples ver que a restrição (3.3) é satisfeita. De fato, quando j se ajusta com i, α_j e $\sum_{S:i\in S,v\notin S}\theta_{S,j}$ começam a crescer à mesma razão. Da mesma forma, a restrição (3.4) é satisfeita, pois se j se ajusta com r então j congela devido a r estar possivelmente aberto.

Considere uma aresta $e=(u,w)$. Seja $l(j)$ a contribuição de j no lado esquerdo da restrição (3.5) para esta aresta, i.e., $l(j)=\sum_{S:e\in\delta(S),r\notin S}\theta^{(1)}_{S,j}$. Suponha que $c_{ju}\leq c_{jw}$. Assim, j se ajusta primeiro a u depois a w. Considere um ponto p na aresta (u,w) à distância x de u, tal que p é o último ponto nesta aresta com o qual j ficou justo (antes de congelar). Então, afirmamos que $l(j)\leq x$. Isto pode ser constatado da seguinte forma. Antes que o "raio"de j alcance u, este não contribui para o somatório $l(j)$, pois e (que está fixo) não pertence a nenhum conjunto S formado até o momento. Sejam $p_1,p_2,\ldots,p_k$ os pontos (na ordem em que se ajustam a j) entre u e p aos quais j se ajusta antes de alcançar p. Quando o raio de j alcança u, este começa a contribuir com $l(j)$ da seguinte forma: para cada ponto p_i entre u e p, a contribuição máxima da variável $\theta_{S,j}$ é a distância entre p_i e p_{i+1}. Após j ajustar com p_{i+1}, um outro conjunto S é formado, e contribui com no máximo a distância entre p_{i+1} e p_{i+2}, e assim por diante. Portanto fica claro que $l(j)$ tem como valor máximo a distância entre u à p. Assim, a afirmação anterior segue.

Definimos $f(j,x)$ como 1 se j está justo com um ponto p e j não foi congelado no instante de tempo ao qual se ajustou com p. Caso contrário, $f(j,x)=0$. Note que $x=\int_0^{c_e}f(j,x)dx$ (aqui usamos a integral, ao invés de um somatório, pois pela Suposição 3.3 (métrica infinitesimal nas arestas), estamos considerando uma aresta ser composta por infinitas arestas de tamanho infinitesimal). Então, como $l(j)\leq x$, podemos dizer que $l(j)\leq\int_0^{c_e}f(j,x)dx$. Assim, a seguinte desigualdade vale para a restrição (3.5):

$$\sum_j\sum_{S:e\in\delta(S),r\notin S}\theta^{(1)}_{S,j}\leq\sum_j\int_0^{c_e}f(j,x)dx\leq\int_0^{c_e}\sum_j f(j,x)dx$$

Para qualquer x, temos que $\sum_j f(j,x)\leq M$, pois, caso contrário, teríamos mais que M demandas justas com um ponto tal que nenhuma dessas demandas estaria congelada – uma contradição. Então $\int_0^{c_e}\sum_j f(j,x)dx$ é no máximo Mc_e, provando o lema. ■

Lema 3.5. *Se i é uma localidade aberta e $j\in D_i$, então $c_{\sigma(j)j}\leq\alpha^{(1)}_j$.*

Demonstração. Como $i\in L'$ então $\sigma(j)=i$. Pela definição de D_i, j está justo com i. Portanto $\alpha^{(1)}_j\geq c_{ij}=c_{\sigma(j)j}$. ■

Lema 3.6. *Ao final da Fase 1, o custo de atribuição (custo de conexão) de qualquer demanda j é no máximo $3\alpha^{(1)}_j$.*

Demonstração. Claramente o lema vale se j está justo com uma localidade em L', pois $\alpha_j\geq c_{ij}$. Caso contrário, suponha que j foi atribuída a $i\in L'$. Seja $i'\in L\setminus L'$ a facilidade

possivelmente aberta que fez com que j congelasse. De acordo com o algoritmo, i e i' são dependentes. Então existe uma demanda k que está justa com ambas facilidades i e i'. Seja $t_{i'}$ o instante de tempo em que i' ficou possivelmente aberta. Analogamente definimos t_i. Devido a i e i' serem dependentes, mas somente i ter sido aberta, temos $t_i \leq t_{i'}$. Como j está justo com i', então $\alpha_j^{(1)} \geq t_{i'}$.

Pela desigualdade triangular, vale $c_{ij} \leq c_{i'j} + c_{i'i}$ e $c_{i'i} \leq c_{i'k} + c_{ik}$. Combinando estas duas desigualdades, temos $c_{ij} \leq c_{i'j} + c_{i'k} + c_{ik}$, onde

$$c_{i'j} + c_{i'k} + c_{ik} \leq \alpha_j^{(1)} + 2\alpha_k^{(1)}. \tag{3.6}$$

A Figura 3.3 ilustra esta situação.

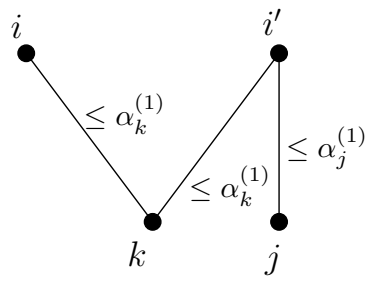

Figura 3.3: A demanda j está justa com i', mas foi atribuída a i. Assim sendo, facilidades i e i' são dependentes, onde k é a demanda em comum entre i e i'.

Afirmamos que $\alpha_k^{(1)} \leq t_{i'}$, decorrente da seguinte argumentação. Sabemos que no instante de tempo $t = \alpha_k^{(1)}$, k está justo com ambos i e i'. Vamos supor que k fica justo antes à i e i está possivelmente aberto. Neste caso, k congela e nunca irá se ajustar a i'. Analogamente, se k fica justo antes à i' e i' está possivelmente aberto, então k congela e nunca irá se ajustar a i. Portanto, obviamente, $\alpha_k^{(1)}$ pára de crescer somente depois de ficar justo com ambos. A partir do exato instante em que k torna-se justo com i e i', podemos ter três casos:

(i) Caso k congele devido a alguma outra facilidade que se tornou possivelmente aberta, antes de i ou i' tornarem-se possivelmente aberta, então $\alpha_k^{(1)} < t_i$ e $\alpha_k^{(1)} < t_{i'}$, e substituindo na Equação 3.6 e usando o fato que $\alpha_j^{(1)} \geq t_{i'}$, o resultado do lema segue.

(ii) Caso k congele devido a i tornar-se possivelmente aberto, então $\alpha_k^{(1)} = t_i$ e como $t_i \leq t_{i'}$, então $\alpha_k^{(1)} \leq t_{i'}$ e substituindo na Equação 3.6 e usando o fato que $\alpha_j^{(1)} \geq t_{i'}$, o resultado do lema segue.

(iii) Caso k congele devido a i' tornar-se possivelmente aberto, então $\alpha_k^{(1)} = t_{i'}$ e $\alpha_k^{(1)} = t_i$ pois $t_i \leq t_{i'}$. Em outras palavras, é impossível i' tornar-se possivelmente aberta

antes de i, no máximo eles ficarão possivelmente abertos no mesmo instante de tempo. Substituindo na Equação 3.6 e usando o fato que $\alpha_j^{(1)} \geq t_{i'}$, o resultado do lema segue.

■

Iremos agora limitar o custo da árvore T. Lembrando que $D' = \bigcup_{i \in L'-\{r\}} D_i$.

Lema 3.7. *val*$(T) \leq 2 \cdot \sum_{j \in D'} \alpha_j^{(2)}$.

Demonstração. Estamos considerando agora a Fase 2. Seja S um conjunto minimal violado, em algum instante qualquer de tempo. Então definimos a variável θ_S como o somatório $\sum_{j \in S} \theta_{S,j}$. Observamos que θ_S cresce à razão 1. Portanto, a Fase 2 simula o algoritmo primal-dual para a árvore enraizada de Steiner, tendo r como raiz. Assim, como afirmamos no Teorema 2.8 do Capítulo 2, o custo da árvore é limitado por $2 \cdot \sum_S \theta_S$, (ver também [14, 1, 37, 9]) onde esta soma é sobre todos subconjuntos de vértices S. Mas, de acordo com a segunda fase do algoritmo, os α_j's crescem à mesma razão que os $\theta_{S,j}$ e portanto $\sum_S \theta_S = \sum_{j \in D'} \alpha_j^{(2)}$. ■

Lema 3.8. *Seja j uma demanda. Se $i \neq v$ então $\alpha_j^{(2)} \leq c_{\sigma(j)j} + c_{ij} + \sum_{S \subseteq V : i \in S, v \notin S} \theta_{S,j}$. Além disso, $\alpha_j^{(2)} \leq c_{\sigma(j)j} + c_{rj}$.*

Demonstração. Se $j \notin D', \alpha_j^{(2)} = \theta_{S,j}^{(2)} = 0$ e as desigualdades acima valem. Fixe uma demanda $j \in D'$ e uma facilidade $i \neq v$. Durante a execução da Fase 2, seja S_t o componente ao qual j contribui no instante de tempo t. Seja t' o menor instante de tempo em que $i \in S_{t'}$, ou, em outras palavras, o instante em que j começa contribuir com i. Após o instante t', ambos lados esquerdo e direito da restrição (3.3) crescem à mesma razão, então precisamos somente limitar o quanto α_j cresceu antes do instante de tempo t'.

Seja $l = \sigma(j)$. Como nós estamos crescendo α_j, então $j \in D_l$ e portanto $c_{lj} \leq \alpha_j^{(1)}$. Afirmamos que $t' \leq M \cdot c_{li}$. Isto vale pois S_t sempre contém l e portanto no instante $t = M \cdot c_{li}$ todas arestas do menor caminho entre l e i já se ajustaram.

Como α_j cresce à razão máxima de $1/M$ (pois há pelo menos M demandas em D_l), então até o instante t', α_j cresceu no máximo $\frac{1}{M} \cdot M c_{li} = c_{li}$. Mas sabemos que $c_{li} \leq c_{lj} + c_{ij}$. Isto prova a primeira desigualdade. A segunda desigualdade é provada de maneira similar. ■

O Lema 3.8 será utilizado para mostrar que os valores das variáveis são viáveis para o programa linear dual, descrito a seguir.

Seja $\alpha'_j = \max(\alpha_j^{(2)} - c_{\sigma(j)j}, 0)$.

Lema 3.9. *A solução dual* $(\alpha', \theta^{(2)})$ *é viável.*

Demonstração. Claramente, os valores das variáveis $\theta^{(2)}$ satisfazem a restrição (3.5), então, pelo Lema 3.8, a solução $(\alpha', \theta^{(2)})$ é uma solução dual viável. ■

Podemos agora provar o teorema principal. Seja O a solução encontrada pelo algoritmo e O^* uma solução ótima. Além disso, seja $\text{val}(O)$ e $\text{val}(O^*)$ os custos respectivos destas soluções.

Teorema 3.10. *O algoritmo proposto fornece uma solução de custo no máximo* $5 \cdot val(O^*)$.

Demonstração. Observe, pela definição de α'_j, que $\alpha_j^{(2)} \leq \alpha'_j + c_{\sigma(j)j}$. Então, pelo Lema 3.7,

$$\text{val}(T) \leq 2\sum_j \alpha'_j + 2\sum_{j \in D'} c_{\sigma(j)j} \leq 2 \cdot \text{val}(O^*) + 2\sum_{j \in D'} \alpha_j^{(1)}.$$

Por outro lado, usando os Lemas 3.5 e 3.6, o custo de atribuição de j é no máximo $\alpha_j^{(1)}$ se $j \in D'$, e no máximo $3\alpha_j^{(1)}$ caso contrário. Assim

$$\begin{aligned}
\text{val}(O) &= \text{cste}(O) + \text{ccon}(O) \\
&= \text{val}(T) + \text{ccon}(O) \\
&\leq 2 \cdot \text{val}(O^*) + 2\sum_{j \in D'} \alpha_j^{(1)} + \sum_{j \in D'} \alpha_j^{(1)} + \sum_{j \notin D'} 3\alpha_j^{(1)}.
\end{aligned}$$

Manipulando os somatórios, temos

$$\begin{aligned}
\text{val}(O) &\leq 2 \cdot \text{val}(O^*) + 3\sum_{j \in D'} \alpha_j^{(1)} + 3\sum_{j \notin D'} \alpha_j^{(1)} \\
&= 2 \cdot \text{val}(O^*) + 3\sum_{j \in D} \alpha_j^{(1)} \\
&\leq 5 \cdot \text{val}(O^*).
\end{aligned}$$

■

Podemos melhorar o fator de aproximação, utilizando qualquer algoritmo ρ_{ST}-aproximado na Fase 2 para construir a árvore de Steiner nas localidades abertas e então conseguir um fator de aproximação de $3 + \rho_{ST}$. Assumimos que $\rho_{ST} \leq 2$.

Teorema 3.11. *Existe um algoritmo* 4.55*-aproximado para o problema Rent-or-Buy.*

Demonstração. Sejam $\text{ccon}(O^*)$ e $\text{cste}(O^*)$ respectivamente o custo de conexão (atribuição) e custo da árvore de Steiner na solução ótima O^*. Usando a árvore ótima em O^*, iremos aumentar esta para construir uma outra árvore nas localidades abertas pelo

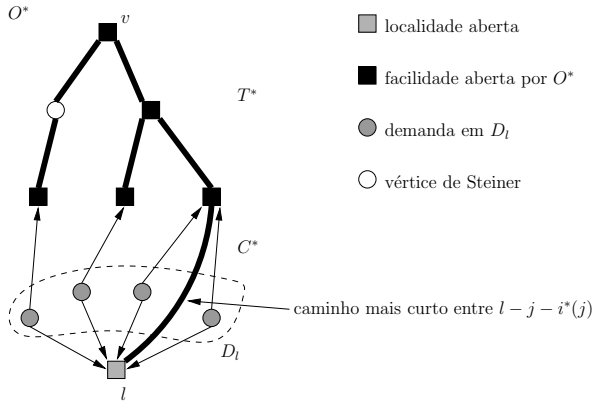

Figura 3.4: Extendendo a árvore em O^* a uma árvore de Steiner nas localidades abertas. T^* são as arestas da árvore de Steiner da solução ótima O^* e C^* são as arestas da solução ótima O^* utilizadas para conectar às demandas à árvore de Steiner.

algoritmo, cujo custo iremos limitar a seguir. Para isto, iremos conectar cada $l \in L'$ à árvore ótima da seguinte maneira. Para cada $j \in D_l$, há um caminho mínimo que liga l à j e j à $i^*(j)$, denotado por $l - j - i^*(j)$, onde $i^*(j)$ é a facilidade a qual j é atribuído em O^* (veja Figura 3.4). Seja $\mathcal{P}_l$ o menor caminho entre todos os caminhos $l - j - i^*(j)$ considerando todos $j \in D_l$. Finalmente, conectamos cada $l \in L'$ à árvore em O^* através de $\mathcal{P}_l$.

Para qualquer $l \in L'$, o custo de adicionar as arestas de conexão é

$$M \cdot \text{val}(\mathcal{P}_l) \leq \sum_{j \in D_l} (c_{i^*(j)j} + c_{lj}),$$

pois $|D_l| \geq M$. Somando sobre todos $l \in L'$, o custo desta árvore aumentada obtida a partir da árvore ótima é no máximo

$$\begin{aligned}
&\text{cste}(O^*) + \sum_{l \in L'} \sum_{j \in D_l} (c_{i^*(j)j}) + \sum_{l \in L'} \sum_{j \in D_l} (c_{lj}) \\
&= \text{cste}(O^*) + \sum_{j \in D'} (c_{i^*(j)j}) + \sum_{j \in D'} c_{\sigma(j)j} \\
&\leq \text{cste}(O^*) + \sum_{j \in D} (c_{i^*(j)j}) + \sum_{j \in D'} c_{\sigma(j)j} \\
&\leq \text{cste}(O^*) + \text{ccon}(O^*) + \sum_{j \in D'} c_{\sigma(j)j}.
\end{aligned}$$

Seja O a solução obtida pelo algoritmo. O procedimento de aumentar a árvore ótima descrito acima assume que sabemos uma árvore ótima. Como não sabemos encontrar

uma árvore ótima eficientemente, podemos utilizar uma ρ_{ST}-aproximação, e assim multiplicamos todo o valor da árvore construída por ρ_{ST}. Portanto, usando os Lemas 3.5 e 3.6, podemos limitar o custo total de nossa solução por

$$\begin{aligned}
\text{val}(O) &= \text{ccon}(O) + \text{cste}(O) \\
&\leq \sum_j c_{\sigma(j)j} + \rho_{ST}(\text{ccon}(O^*) + \text{cste}(O^*) + \sum_{j \in D'} c_{\sigma(j)j}) \\
&= \sum_{j \notin D'} c_{\sigma(j)j} + \sum_{j \in D'} c_{\sigma(j)j} + \rho_{ST}(\sum_{j \in D'} c_{\sigma(j)j}) + \rho_{ST}(\text{ccon}(O^*) + \text{cste}(O^*)) \\
&\leq 3 \sum_{j \notin D'} \alpha_j^{(1)} + 3 \sum_{j \in D'} \alpha_j^{(1)} + \rho_{ST}(\text{val}(O^*)) \\
&= 3 \sum_j \alpha_j^{(1)} + \rho_{ST}(\text{val}(O^*)) \\
&\leq 3(\text{val}(O^*)) + \rho_{ST}(\text{val}(O^*)) \\
&= (3 + \rho_{ST}) \cdot \text{val}(O^*).
\end{aligned}$$

Usando $\rho_{ST} = 1.55$ [33], o resultado segue.

■

3.1.4 Abandonando a suposição

A solução encontrada pode ser inviável pois uma localidade não-vértice pode ter sido aberta como facilidade. O seguinte teorema mostra como contornar este problema.

Teorema 3.12. *Seja a solução k-aproximada obtida pelo algoritmo quando utilizada a Suposição 3.3 (métrica infinitesimal nas arestas). Podemos modificar a solução, abandonando a suposição, sem alterar o fator de aproximação.*

Demonstração. Seja $e = (u, w)$ uma aresta e suponha que abrimos localidades nos pontos internos de e. Seja D_e o conjunto de demandas as quais estão atribuídas a essas localidades. Seja $D_u \subseteq D_e$ o conjunto de demandas que alcançam a sua localidade a que foram atribuídas em e através de u, i.e., $c_{\sigma(j)j} = c_{uj} + c_{\sigma(j)u}$ para $j \in D_u$. Analogamente, definimos D_w. A árvore de Steiner T contém pelo menos u ou w, pois são os únicos caminhos possíveis para se chegar à localidade. Se ambos $u, w \in T$, atribuímos clientes em D_u para u e clientes em D_w para w sem aumentar o custo da solução. Suponha que $u \in T, w \notin T$. Atribuímos todas as demandas em D_u para u. Se $|D_w| < M$, atribuímos clientes em D_w para u e removemos as arestas em T que estão ao longo de e (lembrando que estas arestas removidas têm seu custo multiplicado por M). Caso contrário, atribuímos todos clientes em D_w para w e adicionamos a aresta e inteira em T. Note que T ainda é uma árvore

de Steiner nas localidades abertas. É fácil ver que o custo total só diminui. Portanto, podemos deslocar todas localidades abertas para um vértice de G sem aumentar o custo total da solução. ■

3.2 Connected Facility Location

O CFL pode ser formulado naturalmente como um programa linear inteiro. Por ser conveniente, iremos assumir que $f_r = 0$. Claramente isto não afeta o fator de aproximação do algoritmo, como mostra a seguinte Proposição 3.13.

Proposição 3.13. *Sejam O^* e O uma solução ótima e uma solução obtida por um algoritmo de aproximação, respectivamente, para o problema CFL. Seja v uma facilidade que sabemos fazer parte de ambas soluções O^* e O. Então, assumir $f_v = 0$ não altera o fator de aproximação do algoritmo de aproximação proposto.*

Demonstração. É facil ver que

$$\frac{\text{val}(O) + f_v}{\text{val}(O^*) + f_v} \leq \frac{\text{val}(O)}{\text{val}(O^*)}.$$

onde o lado esquerdo representa as soluções considerando o valor f_v e o lado direito representa as soluções com $f_v = 0$. Se obtivermos um algoritmo α-aproximado para o caso em que $f_v = 0$, então temos que

$$\frac{\text{val}(O)}{\text{val}(O^*)} \leq \alpha,$$

mas combinando com a primeira desigualdade, temos que

$$\frac{\text{val}(O) + f_v}{\text{val}(O^*) + f_v} \leq \alpha,$$

e a proposição segue. ■

Relembrando que i indexa as facilidades em F e j as demandas em D. Podemos agora escrever o programa linear inteiro (PI) para o CFL:

$$\min \sum_{i \in F} f_i y_i + \sum_{j \in D} \sum_{i \in F} c_{ij} x_{ij} + M \sum_{e \in E} c_e z_e \tag{PI}$$

$$\begin{aligned}
\text{s.a.} \sum_{i \in F} x_{ij} &\geq 1 && \forall j \in D, \\
x_{ij} &\leq y_i && \forall i \in F, \forall j \in D, \\
y_v &= 1, && \\
\sum_{i \in S} x_{ij} &\leq \sum_{e \in \delta(S)} z_e && \forall S \subseteq V, r \notin S, \forall j \in D, \\
x_{ij}, y_i, z_e &\in \{0,1\}. && (3.7)
\end{aligned}$$

Onde y_i indica se a facilidade i está aberta, x_{ij} indica se o cliente j está conectado à facilidade i e z_e indica se a aresta e pertence à arvore de Steiner. Ao relaxar as restrições de integralidade (3.7), temos $x_{ij}, y_i, z_e \geq 0$, resultando no programa linear (PL). O dual do programa fica:

$$\max \sum_{j \in D} \alpha_j - \sum_{j \in D} \beta_{vj} \tag{DI}$$

$$\begin{aligned}
\text{s.a.} \quad \alpha_j &\leq c_{ij} + \beta_{ij} + \sum_{S \subseteq V : i \in S, v \notin S} \theta_{S,j} && \forall i \in F, \forall j \in D, i \neq r, && (3.8) \\
\alpha_j &\leq c_{rj} + \beta_{rj} && \forall j \in D, && (3.9) \\
\sum_{j \in D} \beta_{ij} &\leq f_i && \forall i \in F, i \neq r, && \\
\sum_{j \in D} \sum_{S \subseteq V : e \in \delta(S), r \notin S} \theta_{S,j} &\leq M \cdot c_e && \forall e \in E, && \\
\alpha_j, \beta_{ij}, \theta_{S,j} &\geq 0. && &&
\end{aligned}$$

3.2.1 Visão geral do algoritmo

A idéia central é parecida com a do algoritmo da Seção 3.1. Ainda queremos agrupar pelo menos M demandas em cada facilidade aberta, fazendo com que o custo de conectar esta facilidade às outras facilidades abertas (utilizando arestas da árvore de Steiner) possa

ser amortizada pelas demandas agrupadas. No entanto, no CFL, não podemos abrir facilidades em qualquer ponto do grafo. Além disso, uma facilidade tem um custo de abertura que também precisa ser pago para que esta se torne aberta.

No algoritmo que apresentaremos, não será sempre possível agrupar M demandas em uma facilidade que foi aberta. No entanto, conseguiremos garantir que pelo menos M demandas estão agrupadas numa localidade "próxima" desta facilidade aberta (veja Figura 3.5). Na Fase 1 do algoritmo, iremos abrir as facilidades e atribuir cada cliente a alguma facilidade aberta. Além disso, para cada facilidade aberta, iremos conectá-la, utilizando arestas de Steiner, ao ponto próximo onde estão agrupadas pelo menos M demandas. Iremos mostrar que o custo de comprar este caminho é pago pela soma das variáveis duais das demandas agrupadas.

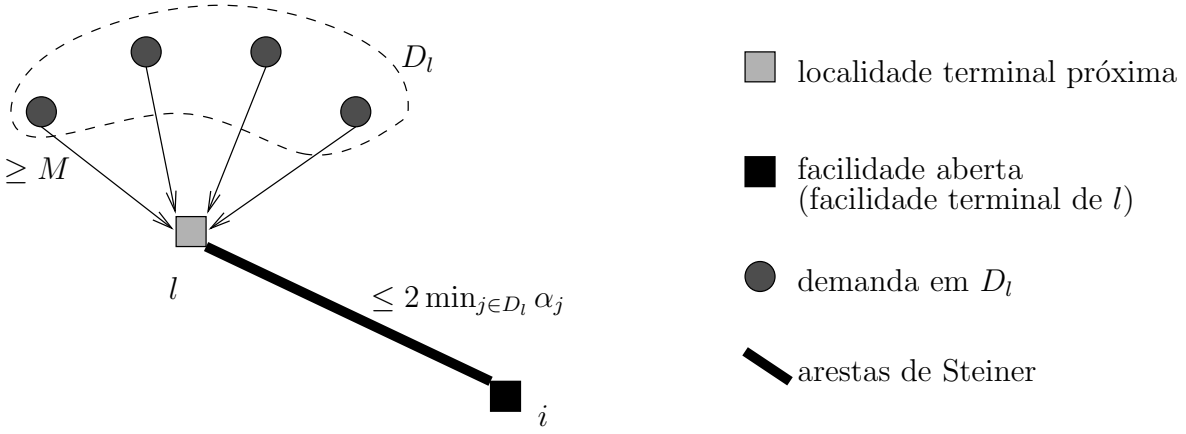

Figura 3.5: Arestas de Steiner conectando uma facilidade aberta ao ponto "próximo"onde M demandas estão agrupadas

3.2.2 Algoritmo

Fase 1

Assim como no algoritmo da Seção 3.1, uma localidade pode ser tanto um vértice em V como ou um ponto ao longo da aresta. As facilidades podem ser abertas somente em localidades em $F \subseteq V$.

Inicialmente, todas variáveis duais são iguais a 0 e somente a facilidade r é possivelmente aberta. Também dizemos que r é uma *localidade terminal*. Lembrando que uma demanda j é dita estar *justa* com uma localidade i se $\alpha_i \geq c_{ij}$.Assim como na Seção 3.1, iremos crescendo cada variável dual α_j até j ficar justo com uma localidade, que chamaremos de localidade terminal, a qual pelo menos M demandas estão justas. No entanto, quando isto acontecer, não congelamos j ainda. Como temos que atribuir j a

uma facilidade aberta e também temos que pagar para abrir facilidades, continuamos a aumentar α_j até j ficar justa com uma facilidade possivelmente aberta. Enquanto isso, se j fica justa com uma facilidade que ainda não está aberta, então j começa a contribuir com pagamento do custo de abertura desta facilidade.

Para descrever o processo primal-dual em detalhes, iremos definir alguns conceitos adicionais. Como antes, uma demanda pode estar congelada ou não-congelada. Além disso, uma demanda j pode estar *livre* ou pode ser uma *escrava*. No tempo $t = 0$, cada demanda j está livre e não-congelada. Dizemos que uma demanda j está *atada* a uma localidade l se j está justo com l e *estava livre quando ficou justa com* l. O *peso* de uma localidade l é o número de demandas que estão *atadas* à l. Dizemos que uma facilidade i foi paga se $\sum_j \beta_{ij} = f_i$.

Em qualquer instante de tempo, definimos S_j como sendo o conjunto total de vértices ao qual uma demanda j está justa. Quando j fica justa com uma facilidade i, temos duas opções: podemos aumentar β_{ij} ou aumentar $\theta_{S_j,j}$. Aumentamos $\theta_{S_j,j}$[1] à mesma razão e continuamos até j ficar justa com uma localidade terminal, ou seja, uma localidade que tem pelo menos M demandas atadas a ela. Neste momento, dizemos que j torna-se um *escravo* – não é mais livre. Analogamente, quando j fica justo com uma localidade l que *não* é uma facilidade, podemos ou não aumentar $\theta_{S_j,j}$ (temos esta opção pois a restrição (3.8) diz respeito às facilidades i). Primeiramente aumentamos $\theta_{S_j,j}$ até j ficar justa com uma localidade terminal e ser declarada como escrava. Após isso, começamos a crescer β_{ij} para cada facilidade $i \in S_j$ e paramos de crescer $\theta_{S_j,j}$. Em outras palavras, aumentamos o α_j de *toda* demanda não-congelada, seja livre ou escrava, a taxa unitária até um dos seguintes eventos acontecer:

1. **O peso de alguma localidade l fica pelo menos M**: l torna-se uma localidade terminal. Se j está livre e justo com l, então agora torna-se *escravo*. De agora em diante aumentamos somente os β_{ij} das facilidades i em S_j (pode não haver nenhuma se o α_j atual é menor que $\min_i c_{ij}$) como descrito acima.

2. **Um j livre fica justo com uma localidade terminal l**: j torna-se escravo. Se $l = r$, conecte j à l e congele l. Caso contrário, paramos de crescer $\theta_{S_j,j}$ e crescemos β_{ij} para facilidades i em S_j.

3. **Uma facilidade i é paga, i.e., $\sum_j \beta_{ij} = f_i$**: i fica possivelmente aberta. Se uma demanda j escrava (não-congelada) está justa com i, conecte j a i e congele j.

4. **Uma demanda j escrava fica justa com uma facilidade i possivelmente aberta**: conecte j a i, congele j.

[1] Poderíamos começar aumentando β_{ij} também, no entanto, algumas variantes de CFL (ver [34]), como o *Connected k-Median* se aproveitam do fato de aumentar primeiro $\theta_{S_j,j}$.

Continuamos este processo até que todas demandas fiquem congeladas. Demanda congeladas não participam de nenhum novo evento. Note que toda demanda j começa livre e não-congelada, então torna-se escrava quando fica justa com uma localidade terminal, e finalmente fica congelada a exatamente uma facilidade possivelmente aberta. Seja $(\alpha^{(1)}, \beta^{(1)}, \theta^{(1)})$ a solução dual obtida. Claramente $\beta_{rj}^{(1)} = 0$ para todo j.

Seja L o conjunto de todas localidades terminais. Seja t_l o instante de tempo em que l foi declarada uma localidade terminal. Seja D_l o conjunto de demandas *atadas* a l. Iremos associar uma *facilidade terminal* a cada $l \in L$. Considere a demanda em D_l com menor $\alpha_j^{(1)}$ e seja i a facilidade possivelmente aberta a qual ela está conectada. Chamamos esta demanda de *demanda representativa* da localidade l, e dizemos que i é a facilidade terminal correspondente a l. A facilidade terminal de r é o próprio r. Seja F' o conjunto de todas as facilidades terminais. Serão abertas somente facilidades que estão no conjunto F'.

Iremos escolher um subconjunto de localidades terminais e abrir as facilidades terminais que correspondem a estas localidades. Para cada localidade l que escolhermos, iremos conectar l à sua facilidade terminal i utilizando arestas de Steiner ao longo do menor caminho entre l e i (veja Figura 3.5). Devemos escolher cuidadosamente o subconjunto de localidades terminais de forma que garanta que uma demanda j não pague para abrir ou conectar mais que uma facilidade.

Dizemos que duas facilidades i, i' são *dependentes* se um dos dois acontece:

(1) há uma demanda j com ambos $\beta_{ij}^{(1)}, \beta_{i'j}^{(1)} > 0$, ou

(2) há uma localidade $l \in L$ e uma demanda j tal que i é uma facilidade terminal correspondente a l, $j \in D_l$ e $\beta_{i'j}^{(1)} > 0$.

A segunda condição foi adicionada para garantir que j não paga para abrir i' e ao mesmo tempo para conectar i a l através de arestas de Steiner.

Dizemos que duas localidades l e l' são *dependentes* se um dos dois acontece:

(1) há uma demanda que está atada tanto a l quanto a l', ou

(2) as facilidades terminais correspondentes a l e l' são dependentes.

Agora, selecionamos de maneira gulosa, um conjunto independente maximal de localidades. Para isso, olhamos as localidades em uma ordem particular descrita a seguir: para cada $l \in L$, associamos um valor ϕ_l. Seja j a demanda representativa de l. Defina $\phi_l = \max(\alpha_j^{(1)}, t_l)$[2], e faça $\phi_r = 0$. Percorremos as localidades em L em ordem de crescimento de ϕ_l, e selecionamos um conjunto independente maximal $L' \subseteq L$. Seja F'' o

[2] $t_l > \alpha_j^{(1)}$ quando j está congelado, e atado a uma localidade l', onde $t_{l'} \leq t_l$

conjuntos de facilidades terminais correspondentes às localidades em L'. Devido à nossa construção de L', as facilidades em F'' são independentes. Abrimos todas as facilidades em F''. Note que $r \in F''$.

Associamos uma localidade terminal $\sigma(j)$ a cada demanda j. Se $j \in D_l$ onde $l \in L'$, faça $\sigma(j) = l$. Note que $\sigma(j)$ é bem-definido (não há mais que um $\sigma(j)$ para um dado j) devido à construção do conjunto independente. Caso contrário, seja l a localidade em L que fez com que j se tornasse escravo. Há uma localidade $l \in L'$ selecionada previamente tal que l e l' são dependentes. Faça $\sigma(j) = l'$. Uma demanda j é atribuída à facilidade $i(j) \in F''$ como segue: se há uma facilidade $i \in F''$ tal que $\beta_{ij}^{(1)} > 0$, atribua j a i. Caso contrário, atribua j à facilidade terminal correspondente à $\sigma(j)$.

Seja $D' = \bigcup_{l \in L' - \{v\}} D_l$. Ligamos por aresta de Steiner cada $l \in L'$ à sua facilidade terminal através do caminho mais curto, formando assim vários componentes. Caso haja ciclos, remova arestas até não haver mais ciclos. Seja T' o conjunto de arestas adicionadas.

Fase 2

Esta fase é análoga à segunda fase do algoritmo da Seção 3.1. Como antes, G é aumentado para incluir arestas incidentes às localidades $l \in L'$. Inicializamos nossos conjuntos minimais violados como os componentes de T'. Todas variáveis duais são inicialmente 0. Nesta fase, não crescemos nenhuma variável β_{ij}. Iremos crescer somente as variáveis α_j das demandas em D'. Para um conjunto S, defina D_S como sendo $\bigcup_{l \in S \cap L'} D_l$. O resto dos passos são idênticos à Fase 2 da Seção 3.1.

A árvore T que iremos construir começa vazia. Para cada CMV S, $j \in D_S$, a variável α_j cresce à razão de $1/|D_S|$. A variável $\theta_{S,j}$ também cresce à mesma razão. Assim, garantimos que $\sum_j \theta_{S,j}$ cresce à razão de 1 para qualquer CMV S. Note que *não* estamos assegurando a viabilidade das restrições (3.8) e (3.9), pois α_j pode ser maior que o lado direito destas restrições.

Crescemos as variáveis duais até uma aresta e se ajustar. Adicionamos e em T e atualizamos os conjuntos minimais violados. Este processo continua até não haver mais conjuntos violados, i.e., continua até termos somente um componente (r está neste componente). Agora, consideramos as arestas de T na ordem inversa em que foram inseridas e removemos qualquer aresta redundante. Esta é nossa solução final.

Assim, temos uma árvore T que conecta todas as facilidades abertas. Seja $(\alpha^{(2)}, 0, \theta^{(2)})$ a solução dual construída nesta fase. Vale observar que é possível que T contenha uma aresta a qual um dos ponto extremos tem uma localidade não-vértice. Isto irá acontecer se tal localidade é uma folha da árvore T. Então, simplesmente removemos tais arestas para obter uma nova árvore que usa somente arestas do grafo original.

3.2.3 Análise

Lema 3.14. *A solução $(\alpha^{(1)}, \beta^{(1)}, \theta^{(1)})$ é uma solução dual viável.*

Demonstração. A prova deste lema é similar à prova do Lema 3.4 e será omitida neste documento. ■

O lema a seguir estabelece um limite máximo no custo do caminho entre uma localidade terminal sua facilidade terminal correspondente.

Lema 3.15. *Seja l uma localidade terminal e i sua facilidade terminal correspondente. Então $c_{il} \leq \min_{j \in D_l} 2(\alpha_j^{(1)} - \beta_{ij}^{(1)}) \leq 2\phi_l$.*

Antes de provar o lema acima, vamos verificar a seguinte proposição, útil para simplificar a prova do lema.

Proposição 3.16. *Seja j uma demanda, e i a facilidade a qual j se conectou. Além disso, $\beta_{ij}^{(1)} > 0$. Seja t_j o instante de tempo em que j tornou-se escravo. Então $\alpha_j^{(1)} = \max(t_j, c_{ij}) + \beta_{ij}^{(1)}$.*

Demonstração. Os valores de $\beta_{ij}^{(1)}$ só começam a crescer quando j vira escravo. Se i está dentro do raio de j de comprimento t_j, então $c_{ij} \leq t_j$ e $\alpha_j^{(1)} = t_j + \beta_{ij}^{(1)}$. Caso contrário, no instante t_j, o raio da demanda j ainda não alcançou i. Então, para o raio continuar crescendo, deve haver pelo menos uma facilidade não possivelmente aberta dentro do raio de tamanho t_j. Assim j vai contribuindo para o pagamento dessa(s) facilidade(s), e ao mesmo tempo, aumentando $\alpha_j^{(1)}$ até alcançar i e começar aumentar $\beta_{ij}^{(1)}$, e então i torna-se possivelmente aberta antes das outras facilidades as quais j está justo. Portanto $c_{ij} \geq t_j$ e $\alpha_j^{(1)} = c_{ij} + \beta_{ij}^{(1)}$. ■

Demonstração do Lema 3.15. Seja j alguma demanda em D_l e k a demanda representativa de l, então k está conectado à i. Então, pela desigualdade triangular, $c_{il} \leq c_{ik} + c_{kl}$. Como k está conectado a i e está atado a l, temos:

$$c_{il} \leq c_{ik} + c_{kl} \leq 2\alpha_k^{(1)} \leq 2\phi_l,$$

provando uma das desigualdades do lema. Para provar a outra desigualdade, temos que analisar dois casos:

(1) Se $\beta_{ij}^{(1)} = 0$ então $c_{il} \leq 2(\alpha_j^{(1)} - \beta_{ij}^{(1)})$, pois $\alpha_j \geq \alpha_k$, e o lema segue.

(2) Caso contrário, se $\beta_{ij} > 0$, seja t_j o instante de tempo em que j tornou-se escravo. Note que $c_{lj} \leq t_j$. Pela desigualdade triangular, temos

$$c_{il} \leq c_{jl} + c_{ij}. \tag{3.10}$$

(i) Se $c_{ij} \leq t_j$ então, pela Proposição 3.16, $t_j = \alpha_j^{(1)} - \beta_{ij}^{(1)}$. Como $c_{lj} \leq t_j$ então $c_{lj} \leq \alpha_j^{(1)} - \beta_{ij}^{(1)}$. Além disso como $c_{ij} \leq t_j$ então $c_{ij} \leq \alpha_j^{(1)} - \beta_{ij}^{(1)}$. Subsituindo c_{lj}, c_{ij} na desigualdade triangular, o lema segue.

(ii) Caso contrário, se $c_{ij} > t_j$ então, pela Proposição 3.16, $c_{ij} = \alpha_j^{(1)} - \beta_{ij}^{(1)}$. Como $c_{lj} \leq t_j \leq c_{ij}$ então $c_{lj} \leq \alpha_j^{(1)} - \beta_{ij}^{(1)}$, e novamente, substituindo os valores na desigualdade triangular, o lema segue.

■

Lema 3.17. *Sejam l e l' localidades terminais dependentes com $\phi_l \leq \phi_{l'}$. Se i é a facilidade terminal correspondente a l, então $c_{il'} \leq 6\phi_{l'}$.*

Demonstração. Seja k a demanda representativa da localidade l. Seja i' a facilidade terminal para l' e k' a demanda representativa de l'. Pelo Lema 3.15, $c_{il} \leq 2\phi_l$ e $c_{i'l'} \leq 2\phi'_l$. Sejam t_i e $t_{i'}$ os instantes de tempo em que i e i' ficam possivelmente abertos respectivamente. Há quatro casos que devem ser considerados, dependendo da razão pela qual l e l' são dependentes.

1. $\exists j \in D_l \cap D_{l'}$. Como j estava livre quando ficou justo com l e l', $c_{lj}, c_{l'j} \leq \max(t_l, t_{l'}) \leq \max(\phi_l, \phi_{l'})$. Pela hipótese do enunciado do lema, $\max(\phi_l, \phi_{l'}) = \phi_{l'}$. Combinando com o Lema 3.15 e desigualdade triangular, temos $c_{il'} \leq c_{il} + c_{ll'} \leq c_{il} + c_{jl} + c_{jl'} \leq 2\phi_l + \phi_{l'} + \phi_{l'} \leq 4\phi_{l'}$.

2. $\exists j$ tal que $\beta_{ij}^{(1)}, \beta_{i'j}^{(1)} > 0$. Isto implica que $c_{i'j}, c_{ij} \leq \alpha_j^{(1)} \leq t_{i'}, t_i$. Portanto $c_{ii'} \leq c_{ij} + c_{i'j} \leq 2\alpha_j^{(1)} \leq 2t_{i'} \leq 2\alpha_{k'}^{(1)} \leq 2\phi_{l'}$, e combinando com o Lema 3.15 e desigualdade triangular, $c_{il'} \leq c_{ii'} + c_{i'l'} \leq 2\phi_{l'} + 2\phi_{l'} = 4\phi_{l'}$.

3. Há uma localidade terminal v (poderia ser l) e demanda $j \in D_v$ tal que i é a facilidade terminal para v e $\beta_{i'j}^{(1)} > 0$. Pelo argumento acima, $c_{i'j} \leq \alpha_j^{(1)} \leq t_{i'} \leq \alpha_{k'} \leq \phi_{l'}$ e combinado com o Lema 3.15 e desigualdade triangular, temos $c_{ij} \leq c_{iv} + c_{jv} \leq 2\phi_v + \alpha_j^{(1)} \leq 3\alpha_j^{(1)}$. Portanto, $c_{ii'} \leq c_{i'j} + c_{ij} \leq \phi_{l'} + 3\alpha_j^{(1)} \leq 4\phi_{l'} \Longrightarrow c_{il'} \leq c_{ii'} + c_{i'l'} \leq 6\phi_{l'}$.

4. Há uma localidade terminal v (poderia ser l'), demanda $j \in D_v$ tal que i' é a facilidade terminal para v e $\beta_{ij}^{(1)} > 0$. Como acima, $c_{ii'} \leq 4\phi_l \Longrightarrow c_{il'} \leq 6\phi_{l'}$.

■

Para uma facilidade aberta i, defina C_i como sendo o conjunto de demandas j tal que $\beta_{ij}^{(1)} > 0$. Seja $C_{F''} = \bigcup_{i \in F''} C_i$. Note que os conjuntos C_i são disjuntos, e todas as demandas em C_i estão atribuídas a i. Relembrando que T' é o conjunto das arestas de Steiner adicionadas na Fase 1.

O próximo lema limita superiormente o custo das arestas em T'.

Lema 3.18. *val*$(T') \leq 2\sum_{j\in D'} \alpha_j^{(1)} - 2\sum_{j\in D'\cap C_{F''}} \beta_{i(j)j}^{(1)}$.

Demonstração. Seja i_l a facilidade terminal correspondente a l. Então $\text{val}(T) \leq \sum_{l\in L'} Mc_{i_l l}$, devido às arestas que formam ciclos que foram removidas (seria uma igualdade caso não houvesse arestas que formam ciclo). Considere qualquer localidade terminal $l \in L'$ com facilidade terminal i correspondente. Pelo Lema 3.15, $c_{il} \leq 2(\alpha_j^{(1)} - \beta_{ij}^{(1)})$ para qualquer $j \in D_l$. Como $|D_l| \geq M$, então

$$Mc_{il} \leq \sum_{j\in D_l} 2(\alpha_j^{(1)} - \beta_{ij}^{(1)}) = 2\sum_{j\in D_l} \alpha_j^{(1)} - 2\sum_{j\in D_l\cap C_{F''}} \beta_{i(j)j}^{(1)}$$

pois $\beta_{ij}^{(1)} > 0$ implica em $j \in C_{F''}$, e $i(j) = i$ para $j \in D_l$ devido a construção do conjunto independente. Somando sobre todos os $l \in L'$, o lema segue. ■

Lema 3.19. *A solução obtida satisfaz* $7\sum_{i\in F''} f_i + \sum_j c_{i(j)j} + \textit{val}(T') + 2\sum_{j\in D'} c_{\sigma(j)j} \leq 7\sum_j \alpha_j^{(1)}$.

Demonstração. Iremos cobrar de cada j uma quantidade cobrar(j) tal que

$$7\sum_{i\in F''} f_i + \sum_j c_{i(j)j} + \text{val}(T') + 2\sum_{j\in D'} c_{\sigma(j)j} \leq \sum_j \text{cobrar}(j) \tag{3.11}$$

$$\leq 7\sum_j \alpha_j^{(1)}. \tag{3.12}$$

Onde

$$\text{cobrar}(j) = \begin{cases} c_{i(j)j} + 7\beta_{i(j)j}^{(1)} & \text{se } j \in C_{F''} - D' \\ c_{i(j)j} + 7\beta_{i(j)j}^{(1)} + 2(\alpha_j^{(1)} - \beta_{i(j)j}^{(1)}) + 2c_{\sigma(j)j} & \text{se } j \in C_{F''} \cap D' \\ c_{i(j)j} + 2\alpha_j^{(1)} + 2c_{\sigma(j)j} & \text{se } j \in D' - C_{F''} \\ c_{i(j)j} & \text{se } j \notin D' \cup C_{F''} \end{cases}$$

Substituindo os valores cobrar(j) na desigualdade (3.11), ficamos com

$$\begin{aligned} &7\sum_{i\in F''} f_i + \sum_j c_{i(j)j} + \text{val}(T') + 2\sum_{j\in D'} c_{\sigma(j)j} \qquad (3.13)\\ &\leq 7\sum_{j\in C_{F''}} \beta_{i(j)j}^{(1)} + \sum_j c_{i(j)j} + 2\sum_{j\in D'} \alpha_j^{(1)} - 2\sum_{j\in D'\cap C_{F''}} \beta_{i(j)j}^{(1)} + 2\sum_{j\in D'} c_{\sigma(j)j} \\ &\leq 7\sum_j \alpha_j^{(1)}. \end{aligned}$$

A desigualdade (3.13) segue diretamente do Lema 3.18 e do fato que, para cada $i \in F''$, todos $j \in C_i$ são atribuídos a i e $\sum_{i \in C_i} \beta_{ij}^{(1)} = f_i$.

Para provar a desigualdade (3.12), note que se $j \in C_{F''}$ então $c_{i(j)j} + \beta_{i(j)j}^{(1)} \leq \alpha_j^{(1)}$. Se $j \in D'$ então $c_{\sigma(j)j} \leq t_{\sigma(j)j} \leq \alpha_j^{(1)} - \beta_{i(j)j}^{(1)}$, pelo Lema 3.15. Além disso, novamente pelo Lema 3.15, se $j \in D' - C_{F''}$ então $c_{i(j)j} \leq 3\alpha_j^{(1)}$. Então, no caso em que $j \in D' \cup C_{F''}$, cobrar$(j) \leq 7\alpha_j^{(1)}$.

Considere o caso onde $j \notin D' \cup C_{F''}$. Iremos mostrar que $c_{i(j)j} \leq 7\alpha_j^{(1)}$. Seja $l' \in L - L'$ a localidade que fez com que j se tornasse escravo e seja $\sigma(j) = l \in L'$. Claramente $\alpha_j^{(1)} \geq t_{l'}$ e como $j \in D_{l'}$ então $\alpha_j^{(1)} \geq \phi_{l'}$. Como $\sigma(j) = l$, l e l' são dependentes com $\phi_l \leq \phi_{l'}$, e $i(j)$ é a facilidade terminal correspondente a l. Portanto, pelo Lema 3.17, $c_{i(j)l'} \leq 6\phi_{l'}$. Isto implica que $c_{i(j)j} \leq 7\alpha_j^{(1)}$. ■

Podemos agora provar o teorema principal. Seja O^* uma solução ótima. e val(O^*) o custo respectivo desta solução.

Teorema 3.20. *O algoritmo descrito fornece uma solução de custo no máximo* $9 \cdot val(O^*)$.

Demonstração. Seja T'' o conjunto das arestas de Steiner adicionadas na Fase 2 do algoritmo. Pelo Lema 3.9, $(\alpha', 0, \theta^{(2)})$ é uma solução dual viável onde $\alpha' = \max(\alpha_j^{(2)} - c_{\sigma(j)j}, 0)$. Portanto, $\text{val}(T'') \leq 2 \cdot \text{val}(O^*) + 2\sum_{j \in D'} c_{\sigma(j)j}$ e o custo da árvore T é no máximo $2 \cdot \text{val}(O^*) + 2\sum_{j \in D'} c_{\sigma(j)j} + \text{val}(T')$. Adicionando isto a $\sum_{i \in F'} f_i + \sum_j c_{i(j)j}$, temos

$$2 \cdot \text{val}(O^*) + 2\sum_{j \in D'} c_{\sigma(j)j} + \text{val}(T') + \sum_{i \in F'} f_i + \sum_j c_{i(j)j},$$

e usando o Lema 3.19, o custo total é no máximo $2 \cdot \text{val}(O^*) + 7\sum_j \alpha_j^{(1)} \leq 9 \cdot \text{val}(O^*)$. ■

Assim como na Seção 3.1, podemos melhorar a aproximação de 9 para $(7+\rho_{ST})$ usando um algoritmo ρ_{ST}-aproximado ($\rho_{ST} \leq 2$) na Fase 2 para construir a árvore de Steiner nos componentes de T'.

Teorema 3.21. *Tomando* $\rho_{ST} = 1.55$ *[33], o algoritmo descrito fornece uma solução de custo no máximo* $8.55 \cdot val(O^*)$.

Demonstração. Sejam $ccon(O^*), cste(O^*)$ os custos de conexão (atribuição) e da árvore de Steiner de uma solução ótima O^*. Podemos obter uma árvore de Steiner nos componentes de T' conectando cada $l \in L'$ à árvore em O^* através de caminho mínimo entre $l - j - i^*(j)$ para $j \in D_l$, onde $i^*(j)$ é a facilidade a qual j está atribuído na solução ótima. Esta árvore tem custo no máximo $cste(O^*) + ccon(O^*) + \sum_{j \in D'} c_{\sigma(j)j}$. Portanto, utilizando o Lema 3.19, o custo total é no máximo

$$\sum_{i \in F''} f_i + \sum_j c_{i(j)j} + \text{val}(T') + \rho_{ST} \sum_{j \in D'} c_{\sigma(j)j} + \rho_{ST}(cste(O^*) + ccon(O^*)) \leq (7 + \rho_{ST} \cdot \text{val}(O^*)).$$

■

3.2.4 Demandas com valor arbitrário

Vamos supor que, ao invés de demandas unitárias, cada cliente j tem demanda $d_j \geq 0$. Todos os resultados mostrados se extendem a este caso. Uma maneira simples de conseguir isso, é fazer d_j cópias do cliente j. Mas isso só funciona se as demandas forem inteiras ou racionais, e mesmo assim, o algoritmo fica com complexidade de tempo pseudo-polinomial. Podemos, contudo, simular esta redução. Na Fase 1, aumentamos cada α_j a velocidade d_j. Então, modificamos as definições de alcance (ficar justo), peso de uma localidade, ϕ_l para um localidade terminal l, e facilidade terminal l, para refletir isso – substituímos α_j por α_j/d_j. Portanto j alcança i se $\alpha_j/d_j \geq c_{ij}$, peso$(i) = \sum_{j:\alpha_j \geq d_j c_{ij}} d_j$ e no caso geral novamente consideramos somente as demandas j atadas a i. No caso geral, para uma localidade terminal l, seja k a demanda atada a l com o menor valor de $\alpha_j^{(1)}/d_j$. Fazemos $\phi_l = \max(\alpha_k^{(1)}/d_k,\ t_l)$ e a *facilidade terminal* para a localidade l é a facilidade possivelmente aberta a qual k está conectado. Na Fase 2, quando aumentamos α_j e $\theta_{S,j}$, crescemos à velocidade de $\dfrac{d_j}{\sum_{j \in D_s} d_j} \leq \dfrac{d_j}{M}$ tal que θ_S aumente a taxa de 1. Os lemas análogos aos lemas que provamos nas Seções 3.1 e 3.2 são facilmente mostrados serem verdadeiros e a razão de aproximação permanece a mesma.

Capítulo 4

Técnica probabilística

Neste capítulo iremos descrever o uso de algoritmos probabilítiscos para a obtenção de fatores de aproximação. Grosseiramente falando, um algoritmo probabilístico é um algoritmo que utiliza um gerador de bits aleatórios. O comportamente deste algoritmo depende não somente das instâncias, mas também dos valores devolvidos pelo gerador de números aleatórios. Lembrando a definição de algoritmo de aproximação, vista no Capítulo 2, a aproximação que iremos obter é com relação ao valor esperado do algoritmo. Deste modo, nem sempre iremos obter soluções dentro da aproximação esperada. No entanto, ao executar várias vezes o algoritmo, podemos garantir com alta probabilidade que ele nos forneça uma solução dentro da aproximação esperada. Para uma introdução sobre algoritmos probabilísticos, veja [26].

Este capítulo foi baseado em resultados obtidos em [12]. Iremos apresentar uma 3.55-aproximação para o problema Rent-or-Buy e, apesar de ser atualmente o melhor fator de aproximação para o problema, a sua desaleatorização nos leva a um fator de aproximação pior que o fator 4.55 apresentado no Capítulo 3. A desaleatorização do algoritmo deste capítulo será vista no Capítulo 8.

Para este capítulo, iremos assumir como verdadeiras as Suposições 3.1 (demandas unitárias) e 3.2 (raiz na solução).

4.1 Rent-or-Buy

Seja $\text{ccon}(O^*)$, $\text{cste}(O^*)$ o custo de conexão e o custo da árvore de Steiner de alguma solução ótima O^* que abre a facilidade raiz r. Seja $\text{val}(O^*) = \text{ccon}(O^*) + \text{cste}(O^*)$ o custo de O^*, $F^* \subseteq V$ as facilidades abertas em O^*, e T^* a árvore de Steiner construída sobre F^* em O^*.

Vamos agora mostrar o algoritmo de aproximação RorB-Simples para o Rent-or-Buy.

O algoritmo pode ser visto como uma redução probabilística do Rent-or-Buy para o problema de encontrar uma boa árvore de Steiner e, em seguida, construir uma árvore de caminhos mais curtos. Seja ρ_{ST} uma constante de aproximação para o problema de árvore de Steiner. Usaremos $\rho_{ST} = 1.55$, devido aos resultados de Robins e Zelikovsky [33].

Algoritmo 4.1: RorB-Simples

Entrada: grafo $G = (V, E)$, demandas $D \subseteq V$, parâmetro $M > 1$;
Saída: árvore de Steiner T, facilidades F;

C1 Com probabilidade $1/M$, marque cada demanda $j \in D$.
Seja $D' \subseteq D$ o conjunto das demandas marcadas.

C2 Construa uma árvore de Steiner ρ_{ST}-aproximada em $F = D' \cup \{r\}$ e *compre* (i.e. pague custo $M \cdot c_e$) as arestas dessa árvore.
Os elementos de F são chamados *facilidades abertas*.

C3 Conecte cada demanda $j \in D$ à sua facilidade aberta mais próxima $i(j)$ em F.

devolva T, F

Lema 4.1. *O custo esperado da árvore obtida no Passo (C2) é no máximo* $\rho_{ST} \cdot val(O^*)$.

Demonstração. Seja T^* a árvore de Steiner construída sobre F^* numa solução ótima O^*. Iremos demonstrar o lema exibindo uma árvore (aleatória) de Steiner T em F com custo esperado no máximo $\text{val}(O^*)$.

Definimos a árvore de Steiner T em F como sendo a união das arestas de T^* e as arestas dos caminhos mais curtos $j - i^*(j)$ para todo $j \in F \backslash \{r\} \subseteq D$, onde j é atribuído a $i^*(j)$ na solução ótima OPT. O custo de $T^* \subseteq T$ é deterministicamente $\text{cste}(O^*)$ (custo da árvore de Steiner em uma solução ótima). Para uma demanda $j \in D$, o custo de comprar o menor caminho $j - i^*(j)$ é $M \cdot c_{i^*(j),j}$ com probabilidade $1/M$ (se $j \in F$) e 0 caso contrário (se $j \notin F$). No pior caso, todos os menores caminhos $i^*(j)$ comprados tem arestas distintas (i.e., os caminhos não tem arestas em comum), então, pela linearidade da esperança:

$$\text{E}[\text{val}(T)] \leq \text{cste}(O^*) + \sum_{j \in D} \frac{1}{M} \cdot M \cdot c_{i^*(j),j} = \text{cste}(O^*) + \text{ccon}(O^*) \leq \text{val}(O^*).$$

Como a árvore T na verdade foi construída sobre uma árvore ρ_{ST}-aproximada e não sobre a árvore ótima, como estávamos considerando, o custo esperado de comprar a árvore T é no máximo $\rho_{ST} \cdot \text{val}(O^*)$. Seja T' a árvore construída pelo algoritmo. Como construímos

T sobre o conjunto de vértices da solução O^* união com o conjunto de vértices marcados pelo algoritmo, e T' foi construída somente sobre os vértices marcados pelo algoritmo, então $\mathrm{E}[\mathrm{val}(T')] \leq \mathrm{E}[\mathrm{val}(T)] \leq \rho_{ST} \cdot \mathrm{val}(O^*)$, provando o lema. ■

Lema 4.2. *O custo esperado do Passo (C3) é no máximo* $2 \cdot val(O^*)$.

Demonstração. Observamos primeiramente que o custo esperado das conexões feitas no Passo (C3) é independente da árvore de Steiner construída no Passo (C2), pois as facilidades abertas foram decididas no Passo (C1). Em outras palavras, isto significa que podemos calcular o custo de conexão do Passo (C3) sem que o custo da árvore de Steiner interfira em nossa análise. Desta forma, vamos assumir, sem perda de generalidade, que a árvore de Steiner do Passo (C2) é dada pela árvore geradora mínima (abreviamos por AGM) no grafo de distâncias de menor caminho em D.

Agora, o algoritmo está utilizando a heurística da AGM, e vamos redefinir algumas ações do algoritmo de uma forma diferente, mas que é equivalente à apresentada pelo algoritmo. Ao invés de lançar moeda para todas as demandas de uma vez só, o novo algoritmo considera cada demanda individualmente numa dada ordem, e joga uma moeda para cada demanda. Dependendo do resultado, a demanda é (a) marcada, adicionada à F e incorporada a árvore de Steiner atual ou (b) conectada a algum vértice que foi marcado anteriormente em F.

Para decidir a ordem dos vértices, vamos definir dois conjuntos. No começo do passo t (i.e., no t-ésimo lançamento de moeda), seja A_t o conjunto de vértices já considerados previamente pelo algoritmo, e $B_t \subseteq A_t$ os vértices que foram marcados. Inicialmente, $A_1 = B_1 = \{r\}$. No passo t, devemos obter o vértice $v_t \in V \backslash A_t$ que está *mais próximo* à B_t e jogamos uma moeda para este vértice. Com probabilidade $1/M$ (iremos chamar o resultado de "cara") definimos A_{t+1} e B_{t+1} adicionando v_t a ambos os conjuntos A_t e B_t, e atualizamos a árvore de Steiner comprando as arestas do menor caminho entre v_t e o vértice mais próximo em B_t. Se a moeda resultou em "coroa", definimos $A_{t+1} = A_t \cup \{v_t\}$ e $B_{t+1} = B_t$, alugando as arestas do menor caminho entre v_t e o vizinho mais próximo em B_t.

Uma observação importante é que o processo incremental que utilizamos acima para construir a árvore de Steiner T sobre F é exatamente o algoritmo de Prim para a AGM (pois v_t é sempre conectado ao vizinho mais próximo), sendo executado no grafo de distâncias de caminho mais curto entre os vértices de F. Além disso, esse novo processo aleatório definido acima implementa de maneira equivalente os dois primeiros passos do Algoritmo 4.1. Podemos facilmente notar que o custo de conexão do processo descrito acima pode ser maior, mas não é menor que o custo do Algoritmo 4.1.

Iremos agora concluir a prova mostrando que o custo esperado de conexão para esse novo algoritmo é no máximo $2 \cdot \mathrm{val}(O^*)$.

Definimos então X_t uma variável aleatória que denota o custo de *alugar* (atribuir v_t ao vizinho mais próximo a B_t) menos o custo de *comprar* (adicionar v_t à B_t e conectá-lo a árvore de Steiner atual) no passo t do algoritmo.

Seja $l(v_t, B_t)$ o custo do caminho entre v_t e o vértice mais próximo em B_t no instante de tempo t. Dado que os $t-1$ lançamentos de moedas anteriores já ocorreram, e portanto v_t e B_t são deterministicamente conhecidos, então o valor esperado de X_t é

$$E[X_t] = \left(1 - \frac{1}{M}\right) \cdot l(v_t, B_t) - \left(\frac{1}{M}\right) M \cdot l(v_t, B_t).$$

Manipulando a equação acima, temos que $E[X_t] \leq 0$. A desigualdade anterior é válida para as primeiras $t-1$ jogadas de moedas, e portanto vale que

$$E[X_t] \leq 0, \forall t.$$

Seja $X = \sum_i X_i$ o custo de conexão menos o custo de Steiner (neste caso, o custo da AGM) da solução O produzida. Ou, em outras palavras,

$$X = \text{ccon}(O) - \text{cste}(O).$$

Pela linearidade da esperança, temos

$$E[X] = E\left[\sum_i X_i\right] = \sum_i E[X_i] \leq 0.$$

Pela desigualdade acima podemos afirmar que o custo de conexão esperado decorrente do algoritmo incremental é no máximo o custo esperado da AGM em F, pois $E[X] = E[\text{ccon}(O)] - E[\text{cste}(O)] \leq 0$. Reescrevendo, temos

$$E[\text{ccon}(O)] \leq E[\text{cest}(O)].$$

Mas sabemos que utilizando a heurística da AGM, apresentada no Algoritmo 2.3 do Capítulo 2, temos uma 2-aproximação para a árvore de Steiner ótima. Em outras palavras

$$E[\text{ccon}(O)] \leq E[\text{cest}(O)] \leq 2 \cdot E[\text{val}(T^*)],$$

mas, no Lema 4.1 provamos que $E[val(T^*)] \leq val(O^*)$, onde O^* é uma solução ótima, que combinado com as desigualdades acima, conclui a prova do lema. ■

Seja val(O) o custo da solução O obtida pelo Algoritmo 4.1.

Teorema 4.3. *O Algoritmo 4.1 é um algoritmo com fator de aproximação* $(2 + \rho_{ST})$ *para o problema Rent-or-Buy.*

Demonstração. O teorema segue diretamente dos Lemas 4.1 e 4.2, que limitam o custo esperado da árvore de Steiner e custo esperado de conexão separadamente. Mais formalmente

$$\begin{aligned} E[\text{val}(O)] &= E[\text{cste}(O)] + E[\text{ccon}(O)] \\ &\leq \rho_{ST} \cdot \text{val}(O^*) + 2 \cdot \text{val}(O^*) \\ &= (2 + \rho_{ST}) \cdot \text{val}(O^*). \end{aligned}$$

■

4.2 Variações

A análise do Algoritmo 4.1 é flexível e permite diversas variações.

1. Um modo simples de permitir demandas com pesos inteiros não uniformes (ou, no caso de pesos racionais, aplicando uma escala), que também será útil nas seções posteriores, é modificar o Passo (C1) para que d_j moedas sejam jogadas para uma demanda j com peso d_j, e esta demanda é marcada se pelo menos uma das moedas lançadas resultarem em "cara". Assim, nós trocamos a demanda j por d_j demandas com peso 1, todas localizadas na mesma posição. Fazer isso é equivalente a jogar uma moeda para j que resulte em "cara" com probabilidade $1 - (1 - 1/M)^{d_j}$. Esse processo pode ser implementado eficientemente mesmo quando o peso das demandas não é limitado polinomialmente.

2. O tempo de execução do algoritmo pode ser melhorado por um fator de $|V|$ escolhendo um vértice raiz r aleatoriamente no conjunto de vértices D. No entanto, utilizar este algoritmo mais rápido aumentaria o fator de aproximação para $\alpha(2+\rho_{ST})$, onde $\alpha = 1 + M/|D|$ é, sem perda de generalidade, no máximo 2. Ao escolhermos r aleatoriamente, este tem probabilidade $1/M$ de pertencer a algum vértice marcado pelo algoritmo e probabilidade $1 - 1/M$ de não pertencer. Seja $\text{val}(A)$ o custo do Algoritmo 4.1 (que sabemos ser $2+\rho_{ST}$-aproximado), e seja $\text{val}(A')$ o custo do algoritmo ao escolher r aleatoriamente. Seja $l(v, S)$ o custo do caminho mínimos entre v e o vértice mais próximo em $S \subseteq D$. Desta forma temos

$$E[\text{val}(A')] = \frac{1}{M} E[\text{val}(A)] + \left(1 - \left(\frac{1}{M}\right)\right) \left(E[\text{val}(A)] - l(r, F) + M \cdot l(r, F)\right).$$

Fazendo manipulações algébricas, obtemos

$$\begin{aligned}E[\text{val}(A')] =& \frac{1}{M}E[\text{val}(A)] + E[\text{val}(A)] - \frac{1}{M}E[\text{val}(A)] \\ &+ \left(1 - \left(\frac{1}{M}\right)\right)(-l(r,F) + M \cdot l(r,F)) \\ =& E[\text{val}(A)] + l(r,F)\left(1 - \left(\frac{1}{M}\right)\right)(M-1)\,.\end{aligned}$$

A esperança do número de demandas que foram marcadas pelo algoritmo é $|D|/M$, ou seja $E[|F|] = |D|/M$. Portanto, se fixarmos o conjunto F, a demanda r pode ser escolhida aleatoriamente dentre $|D| - |D|/M$ demandas. Desta forma $l(r,F)$ é uma variável aleatória sobre r aleatório e F fixo, cuja esperança é

$$E[l(r,F)] = \frac{\sum_{r \in D \setminus F} c_{r,i(r)}}{|D| - |D|/M},$$

onde $i(r) \in F$ é a demanda a qual r foi atribuída. Note que o valor $c_{r,i(r)}$ é determinístico, dado que r e F são fixos. Assim, podemos reescrever $E[\text{val}(A')]$ como

$$\begin{aligned}E[\text{val}(A')] =& E[\text{val}(A)] + \frac{\sum_{r \in D \setminus F} c_{r,i(r)}}{|D| - |D|/M}\left(1 - \left(\frac{1}{M}\right)\right)(M-1) \\ =& E[\text{val}(A)] + \frac{\sum_{r \in D \setminus F} c_{r,i(r)}}{\frac{|D|(M-1)}{M}}\left(1 - \left(\frac{1}{M}\right)\right)(M-1) \\ =& E[\text{val}(A)] + \frac{\sum_{r \in D \setminus F} c_{r,i(r)}}{|D|} M \left(1 - \left(\frac{1}{M}\right)\right) \\ =& E[\text{val}(A)] + \frac{\sum_{r \in D \setminus F} c_{r,i(r)}}{|D|}(M-1) \\ \leq& E[\text{val}(A)] + \frac{M}{|D|}\sum_{r \in D \setminus F} c_{r,i(r)},\end{aligned}$$

e como $\sum_{r \in D \setminus F} c_{r,i(r)} \leq E[\text{val}(A)]$, temos

$$\begin{aligned}E[\text{val}(A')] \leq& E[\text{val}(A)] + \frac{M}{|D|}E[\text{val}(A)] \\ =& \left(1 + M/|D|\right) E[\text{val}(A)].\end{aligned}$$

Como $E[\text{val}(A)]$ é uma $2+\rho_{ST}$-aproximação, então $E[\text{val}(A')]$ é uma $(1+M/|D|)(2+\rho_{ST})$-aproximação.

3. Se as facilidades não podem ser abertas em vértices arbitrários do grafo (ou, equivalentemente, as facilidades tem custo 0 ou ∞) então realocar cada demanda para as potenciais facilidades abertas mais próximas e executar o Algoritmo 4.1 nos fornece um fator de aproximação de $(4 + \rho_{ST})$. Com custos arbitrários nas facilidades, podemos obter um fator de aproximação constante pelo Algoritmo 4.1 se utilizássemos uma árvore-estrela (aproximada) de Steiner [19, 34] ao invés da árvore de Steiner no Passo (C2). A fator de aproximação é um pouco pior que 8.55, apresentado no Capítulo 3, conseguido por Swamy e Kumar [34], e os detalhes dessa redução serão omitidos neste documento.

Parte II

Compartilhamento de custos

Capítulo 5

Introdução ao compartilhamento de custos

Nos capítulos anteriores, estávamos preocupados em "baratear" o preço da rede construída. No entanto, ao tratar de problemas de projeto de redes, uma interpretação mais realista do problema não está somente preocupada com o fato de conseguir o menor preço da rede a ser construída. Além disso, é importante também estabelecer uma maneira dos usuários (demandas, clientes, etc.) dividirem entre si o preço da rede resultante. Em outras palavras, existe uma raiz no sistema que é responsável pela construção da rede. Após essa raiz construir a rede, ela quer dividir o preço final entre os usuários participantes da rede. No entanto, dependendo do modo como é realizada a divisão do custo, esta pode acarretar em alguns problemas.

Neste capítulo, iremos apresentar o conceito do compartilhamento de custos, e sua ligação com a área de algoritmos aproximados. O principal resultado deste capítulo é uma função de cost-sharing "ótima" para o problema da árvore geradora mínima. Este resultado foi obtido por Jain e Vazirani [17] que, além do resultado, prepararam também no mesmo artigo uma introdução ao assunto.

5.1 Motivação

Para exemplificar, vamos considerar o exemplo de compartilhamento de carros (carpooling). Suponha que três empregados de uma firma utilizam o compartilhamento de carros para reduzir o custo de sua viagem diária. O custo de dirigir de um empregado a outro e de um empregado para a firma é dado na Figura 5.1. Na figura, os empregados são denotados pelos números $1, 2$ e 3, e a firma por 0. Uma árvore geradora mínima nesta rede é $\{01, 12, 13\}$ com custo 18. Esta árvore corresponde ao plano de compartilhamento de carros no qual os empregados 2 e 3 dirigem seus carros sozinhos até o empregado 1, e

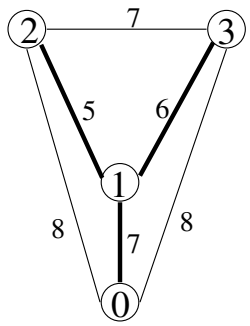

Figura 5.1: O custo de dirigir e uma árvore geradora mínima.

lá todos pegam um único carro para então chegar à firma.

Após resolver o problema de encontrar uma árvore geradora mínima (ver definição do Problema 2.4), os empregados se deparam com o problema de como fazer para dividir o custo de 18 entre eles de maneira justa. É nesta hora que a *teoria dos jogos cooperativos* entra em cena para realizar a divisão dos custos. Os empregados consideram o *jogo da árvore geradora mínima* (N, c), onde $N = \{1, 2, 3\}$ e $c : 2^N \backslash \{\emptyset\} \to \mathbb{R}$ é a função característica que calcula, para cada $S \in 2^N \backslash \{\emptyset\}$ o custo $c(S)$ de uma rede de custo mínimo conectar cada empregado em S à firma. Portanto $c(123) = 18$ e $c(23) = 15$ pois $\{02, 03\}$ é uma árvore geradora mínima para $S = \{2, 3\}$. Uma maneira de dividir o custo $c(N)$ de forma justa é por meio de uma *alocação de núcleo*, que é um vetor $x_i, i \in N$ que é eficiente, ou seja, satisfaz $\sum_{i \in N} x_i = c(N)$ e nenhum subconjunto tem intenção de abandonar o esquema, ou seja, $\sum_{i \in S} x_i \leq c(S), S \subseteq N$ (note que se para algum S, $\sum_{i \in S} x_i > c(S)$ então alguém ganha por estar na solução e o subconjunto S poderia abandonar o esquema todo e montar uma solução independentemente). Bird [3] mostrou como computar um elemento de núcleo de uma árvore geradora mínima. Primeiramente, deve achar a árvore geradora mínima através do algoritmo de Prim, que a cada passo adiciona uma aresta entre um nó que já está conectado com a raiz e um nó que ainda não está conectado. Após isso, a regra de Bird atribui o custo da aresta que foi adicionada à árvore num determinado passo ao vértice que foi conectado à raiz neste mesmo passo. No exemplo da Figura 5.1, o algoritmo de Prim primeiro adiciona a aresta 01, depois 12 e finalmente 13, e a regra de Bird nos fornece a alocação de núcleo $x = (7, 5, 6)$.

Suponha agora que um quarto empregado está querendo se juntar ao esquema de compartilhamento de carros. Os custos de dirigir de 4 para os outros empregados e para a firma são dados na Figura 5.2, assim como a árvore geradora mínima para a nova situação.

A regra de Bird aplicada a esta nova situação nos fornece a alocação $x = (5, 6, 6, 3)$. Nesta nova situação, o empregado 2 deve pagar 6, sendo que na situação antiga ele pagava 5. Portanto, se os empregados utilizarem a regra de Bird para compartilhar o custo, o

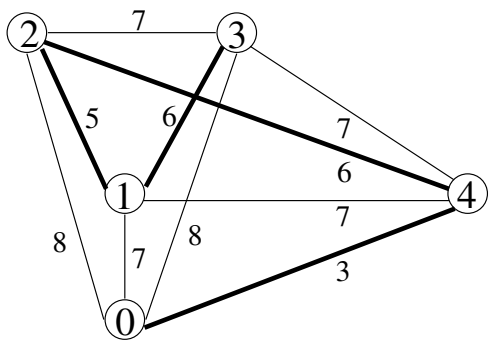

Figura 5.2: O custo de dirigir e uma árvore geradora mínima na nova situação.

empregado 2 irá proibir a entrada do empregado 4. Assim, a regra de Bird não parece ser uma boa estratégia para dividirmos os custos.

A questão central que iremos tratar nos próximos capítulos é se, para os problemas Connected Facility Location e Rent-or-Buy, existe uma maneira justa de dividir os custos entre os clientes e, além disso, garantir que criemos uma alocação onde, ao aumentar a coalisão de participantes, nenhum participante pague a mais. Se conseguirmos garantir esta última propriedade, então temos um *esquema de alocação cross-monotonic*, como veremos mais adiante.

A teoria dos jogos é dividida em jogos cooperativos e jogos não-cooperativos. Nos jogos não-cooperativos, um jogo é um modelo detalhado de todas as ações disponíveis aos jogadores. Em contraste, os jogos cooperativos abstraem esse nível de detalhes e descrevem somente as saídas que resultam quando os jogadores agem em conjunto em diferentes combinações. Ambos assuntos são muito ricos e extensos, neste documento estaremos introduzindo somente os conceitos essenciais sobre jogos cooperativos.

No exemplo do compartilhamento de carros, a rede construída era uma árvore geradora mínima, um problema computacional resolvido em tempo polinomial. Mas nem sempre estaremos diante de problemas fáceis de resolver, motivando a aplicação das técnicas de algoritmos aproximados à teoria dos jogos cooperativos.

Antes de falar sobre o CFL e o R-or-B, vamos formalizar algumas definições sobre teoria dos jogos cooperativos e então resolveremos o problema da árvore geradora mínima. O problema da árvore geradora mínima, mesmo que utilizado como "caixa-preta" pelo CFL e R-or-B, tem um papel importante e central para o entendimento dos conceitos que serão introduzidos e também no desenvolvimento de vários outros algoritmos que resolvem problemas de projeto de redes.

Assim como o exemplo do compartilhamento de carros, existem inúmeros outros problemas de projeto de redes que podem ser resolvidos utilizando a teoria dos jogos. A seguinte definição generaliza essa classe de problemas.

Definição 5.1 (Jogos Cooperativos para Projeto de Redes)**.** Seja r uma companhia provedora de serviço (raiz) e V o conjunto de todos usuários (jogadores) que são possíveis clientes de r. O objetivo de r é selecionar um subconjunto $D \subseteq V$ de usuários, construir uma rede que atenda os usuários em D e dividir o custo desta rede com cada usuário $i \in D$. Cada usuário i tem um atributo de *utilidade* u'_i, que é quanto i está disposto a pagar para receber o serviço de r. Seja x_i o custo-compartilhado (cost-share) que r atribuiu ao usuário i. O *benefício* do usuário i é dado por $u'_i - x_i$. Portanto, um usuário em V pode ou não participar da rede, dependendo se o seu benefício é positivo ou não. Cada usuário age por conta própria, ou seja, para maximizar o benefício, ele pode mentir à r sobre sua utilidade, informando um outro valor, digamos u_i (por exemplo, i pode decidir mentir que sua utilidade é menor para r cobrar menos. No entanto, ao diminuir sua utilidade, i pode deixar de ser atendido por r.).

Como vimos anteriormente, além de construir a rede, temos que compartilhar o custo da rede entre os usuários. O termo em inglês geralmente utilizado para compartilhamento de custo é *cost-sharing*, usaremos esse termo a partir de agora. Um algoritmo para realizar o compartilhamento de custos é composto por um *mecanismo de cost-sharing*, e este por sua vez utiliza uma *função de cost-sharing*. Poderíamos considerar o mecanismo e a função de cost-sharing como uma coisa só, no entanto essa separação existe pois utilizaremos sempre o mesmo mecanismo de cost-sharing, nos restando somente encontrar a função para utilizar com tal mecanismo. Esses detalhes serão vistos mais adiante.

No caso do compartilhamento de custos, as soluções e seus respectivos valores se dão sobre um conjunto de usuários pré-definido, mas temos a necessidade de comparar soluções sobre diferentes conjuntos de usuários. Por exemplo, no problema da árvore de Steiner mínima, temos necessidade de realizar comparação entre soluções com conjuntos diferentes de vértices terminais. Isso ficará mais claro no decorrer do capítulo. Antes, vamos introduzir uma nova notação capaz de expressar essas mudanças. Assim, seja O uma solução obtida sobre o conjunto de usuários D. Denotamos por $\text{val}(O, D)$ o custo da solução O construída sobre o conjunto de usuários D.

A seguir, são apresentadas as definições de mecanismo de cost-sharing e função de cost-sharing.

Definição 5.2 (Mecanismo de Cost-Sharing)**.** Dada uma instância de um problema de projeto de redes, onde V é o conjunto de demandas (usuários), chamamos de *mecanismo de cost-sharing* um algoritmo que

(i) escolhe um conjunto $D \subseteq V$ dos usuários que farão parte da rede e;

(ii) distribui o custo $val(O, D)$ da solução O obtida entre todos os usuários em D tal que cada usuário $i \in D$ possa pagar o custo a si atribuído. Para realizar a distribuição, utiliza-se a *função de cost-sharing*.

Definição 5.3 (Função de Cost-Sharing)**.** Sejam V o conjunto de todos os usuários, e $D \subseteq V$ o conjunto de usuários que participam da rede construída. Uma *função de cost-sharing* $\xi(D, i)$[1] calcula o custo-compartilhado (cost-sharing) que cada usuário $i \in D$ deve pagar. Se $i \in V \backslash D$ então $\xi(D, i) = 0$.

5.2 O modelo multicast

Vamos assumir um modelo no contexto do *roteamento multicast*, que facilita o entendimento e tem ligação direta com o problema da árvore geradora mínima e árvore de Steiner (ver definição dos Problemas 2.4 e 2.5). Em uma rede, a maneira mais simples de fazer o broadcast de uma mensagem para vários receptores é o *unicasting*, onde a mensagem é enviada individualmente para cada receptor. Isto resulta em várias cópias da mesma mensagem nos enlaces (arestas) da rede. Esse desperdício da largura de banda pode ser evitado usando uma forma diferente de roteamento, chamado *roteamento multicast*. O roteamento multicast usa uma árvore que conecta todos os receptores à raiz. A raiz envia uma cópia da mensagem a cada vértice vizinho na árvore. Esses vértices, por sua vez, agem como raízes para os vértices que estão nos níveis abaixo da árvore. Dessa maneira, uma raiz pode alcançar vários receptores sem a necessidade de enviar várias cópias da mesma mensagem em algum enlace.

No roteamento multicast, tradicionalmente é fixada uma árvore contendo a raiz e todos os possíveis receptores. A rede é então restrita a esta árvore somente. Sempre que uma mensagem precisa ser enviada por broadcast para um subconjunto de receptores, o roteamento multicast utiliza uma subárvore desta árvore. Isto não é uma boa idéia quando o conjunto de receptores varia com a mensagem, uma vez que uma subárvore da árvore fixada pode ser arbitrariamente mais custosa que a árvore mais barata que conecta os receptores à raiz (lembrando que a rede completa é um grafo e, ao fixar uma árvore, estamos ignorando as outras arestas do grafo). Então, se fizermos com que a rede não esteja restrita a uma árvore fixada, o problema de encontrar a árvore de multicast mais barata para um dado conjunto de receptores é na verdade o problema da árvore de Steiner (ver definição do Problema 2.5), que sabemos ser NP-difícil. Existem vários algoritmos de aproximação para o problema da árvore de Steiner, no entanto, como visto no exemplo do compartilhamento de carros, não basta aplicarmos um algoritmo com bom fator de aproximação, sem sabermos como compartilhar o custo da solução construída entre os receptores. Assim, dado um problema, buscamos satisfazer algumas propriedades

[1] A rigor, a notação acima deveria referenciar sobre qual solução estamos definindo o cost-sharing. A notação seria então algo como $\xi(O, D, i)$, dado que O é uma solução. Mas a notação apresentada já foi informalmente estabelecida como um padrão na área e de forma geral não há duplas interpretações na notação introduzida.

que garantem um cost-sharing ótimo. Em outras palavras, se tais propriedades forem garantidas, teremos dividido de maneira ótima o custo da rede entre os usuários. Nem sempre todas as propriedades poderão ser satisfeitas para um dado problema, às vezes temos que deixar de satisfazer uma certa propriedade, ou tentar maneiras que aproximem a propriedade de ser satisfeita.

Seja G um grafo não-direcionado com pesos c_e nas arestas, e um vértice r que é a *raiz*, de onde são enviadas as mensagens de broadcast. Todos os outros vértices são *usuários*. Denotamos por V o conjunto de todos usuários. Assumimos também que uma mensagem pode ser duplicada em qualquer vértice sem custo adicional. Para transmitir uma mensagem pela aresta e, paga-se c_e. O custo de fazer broadcast é o custo total de todas as arestas que foram selecionadas. Assumimos que cada aresta tem capacidade infinita e, portanto, uma mensagem pode ser enviada do vértice i ao vértice j através do caminho mínimo entre eles. Além disso, assumimos que os pesos das arestas satisfazem desigualdade triangular.

Agora, supomos que a raiz tem uma mensagem para fazer broadcast e todos usuários i relataram as suas respectivas utilidades u_i para a raiz. As tarefas que devemos realizar são:

1. selecionar um conjunto $D \subseteq V$, que são os usuários que irão receber a mensagem,
2. construir uma árvore T que contém D para fazer o broadcast da mensagem, e
3. para cada usuário i definir um preço x_i, que é o quanto a raiz está cobrando do usuário i por transmitir a mensagem.

Por conveniência de notação iremos também representar D pela função q, i.e., $q_i = 1$ se $i \in D$ e $q_i = 0$ caso contrário.

Existem propriedades computacionais e propriedades econômicas a serem satisfeitas. A única propriedade computacional que estamos considerando neste documento é o tempo polinomial disponível para computação. As propriedades econômicas são listadas abaixo.

1. **Otimalidade**: T é uma solução ótima que conecta todos os usuários em D com a raiz r.
2. **Sem Transferências Positivas (STP)**: para cada usuário i, $x_i \geq 0$, ou seja, os usuários não vão ser pagos para receber o serviço.
3. **Participação Voluntária (PV)**: $q_i u_i - x_i \geq 0$, ou seja, se $i \notin D$ então $x_i = 0$. Se $i \in D$ então $x_i \leq u_i$, i.e., somente os usuários que recebem a mensagem é que pagam. Além disso, eles nunca vão ser cobrados a mais do que suas utilidades relatadas. Em outras palavras, cada usuário tem a opção de não receber a mensagem, sendo então o benefício obtido de 0.

4. **Soberania do Consumidor (SC)**: todo usuário tem garantia de receber o serviço se o valor de sua utilidade for suficientemente grande.

5. **Budget-Balance (BB)**

   (a) Recuperação do custo: $\sum_{i \in D} x_i \geq \text{val}(T, D)$, i.e, os usuários pagam pelo custo do serviço e, portanto, o custo de fazer broadcast da mensagem é recuperado de todos os usuários.

   (b) Competitividade: $\sum_{i \in D} x_i \not> \text{val}(T, D)$, i.e, nenhum excesso de cobrança é criado, pois se houver excesso, algum outro competidor pode oferecer o serviço mais barato.

   A condição *budget-balance* consiste em satisfazer ambas propriedades (a) e (b), ou seja, $\sum_{i \in D} x_i = \text{val}(T, D)$.

6. **Eficiência**: $\sum_{i \in D} u_i - \text{val}(T, D)$ é maximizado, ou seja, o ganho total é o melhor possível. Na verdade queremos que $\sum_{i \in D} u'_i - \text{val}(T, D)$ seja maximizado, mas a próxima propriedade, se obtida, acaba por garantir isso.

7. **Group Strategyproof**: Nenhum usuário ou grupo de usuários obtém vantagens ao mentir sobre sua utilidade u_i. Em outras palavras, mesmo se um grupo de usuários conspirar, a estratégia dominante para cada jogador é relatar o valor verdadeiro de sua utilidade. Para ser mais preciso, considere uma coalizão C de usuários. Seja $u_j = u'_j$ para todo $j \notin C$. Sejam (q, x) e (q', x') os usuários servidos e custos em u e u' respectivamente, quando obedecidas as regras anteriores. Agora, a propriedade *group strategyproof* requer que se a desigualdade

   $$u'_i q_i - x_i \geq u'_i q'_i - x'_i$$

   vale para todo $i \in C$ então deve valer com igualdade para todo $i \in C$ também, i.e., se nenhum membro de C é prejudicado por mentir sobre o valor da sua utilidade então nenhum membro de C é beneficiado também.

Para vários problemas, incluindo o roteamento multicast, a propriedade de Otimalidade não é computacionalmente possível de ser satisfeita eficientemente (i.e., em tempo polinomial), assumindo que P $\neq$ NP. Portanto, esta propriedade será relaxada, fazendo com que a solução encontrada esteja próxima de uma certo fator da solução ótima, para isso utilizamos algoritmos aproximados.

Vários resultados deste capítulo serão somente enunciados, omitindo as suas demonstrações. Isto porque o estudo destes resultados faz parte de uma outra área cuja adesão dos conhecimentos e conceitos iniciais neste documento seria proibitivamente grande, além

de fugir de nosso objetivo, que é somente encontrar uma função de cost-sharing que seja cross-monotonic, como veremos adiante. Vale observar também que poderíamos simplesmente apresentar a definição de função de cost-sharing cross-monotonic, e nos capítulos que seguem tentar obter funções deste tipo, mas neste caso não haveria motivação alguma. O principal objetivo deste capítulo é entender a motivação e compreender porque é interessante obter funções de cost-sharing que sejam cross-monotonic. Assim sendo, tomamos a liberdade de utilizar alguns resultados como "caixa-preta", a fim da exposição da motivação desejada.

O teorema a seguir é um resultado clássico da teoria dos jogos.

Teorema 5.4. *[10, 32] Seja M um mecanismo de cost-sharing que tenha a propriedade* group strategyproof. *Então M não pode ter ambas as propriedades* budget-balance *e* eficiência.

O seguinte teorema, de Feigenbaum, Papadimitriou e Shenker [8], estabelece algumas limitações nas propriedades que devemos satisfazer.

Teorema 5.5. *[8] No problema de roteamento multicast, obter um fator de aproximação constante para a propriedade* eficiência *é NP-difícil.*

Portanto, para os propósitos deste documento, a propriedade de eficiência será abandonada, é impossível satisfazê-la em tempo polinomial de computação, considerando que estamos usando o modelo de máquina de Turing tradicional e P $\neq$ NP. Além disso, como um caso particular do CFL e do R-or-B é a própria árvore de Steiner, que por usa vez é solução do roteamento multicast, então também iremos desconsiderar esta propriedade para estes problemas nos capítulos que seguem.

Moulin e Shenker [28] mostraram que a maioria das propriedades acima podem ser satisfeitas (com excessão das propriedades de otimalidade e budget-Balance) se conseguirmos uma função de cost-sharing que seja *cross-monotonic*, cuja definição apresentamos abaixo.

Definição 5.6. Uma função de cost-sharing ξ é dita ser *cross-monotonic* se para $D \subseteq D'$ então $\xi(D, i) \geq \xi(D', i)$. Em outras palavras, o valor da função de cost-sharing de qualquer usuário não deve aumentar caso outros usuários entrem no sistema.

Além disso, Moulin e Shenker também desenvolveram um mecanismo para ser usado com uma função cross-monotonic, que escolhe o conjunto $D \subseteq V$ e calcula os valores x_i do vetor de custo compartilhado. Esse mecanismo, que chamaremos de *mecanismo M^**, é apresentado abaixo.

Algoritmo 5.1: Mecanismo $M^*(\xi)$

Entrada: função de cost-sharing ξ, rede $G = (V, E)$;
Saída: conjunto $D \subseteq V$, valores x_i do vetor de custo compartilhado;
início
  $D \leftarrow V$;
  repita
    se $u_i < \xi(D, i)$, *para algum* $i \in D$ **então**
      $D \leftarrow D - i$; `// em qualquer ordem de retirada dos i's`
  até $u_i \geq \xi(D, i)$ *para todos usuários* $i \in D$;
  devolva D *e* $x_i = \xi(D, i)$ *para todos* $i \in D$;
fim

Considerando o modelo do roteamento multicast, este mecanismo começa com a tentativa de enviar a mensagem para todos os usuários, e usa ξ para determinar o custo compartilhado de cada usuário. O mecanismo então descarta, em ordem qualquer, os usuários que não puderem pagar por receber a mensagem. As iterações se seguem até que o grupo de usuários possa pagar os custos-distribuídos determinados. Devido ao fato de ξ ser cross-monotonic, o eventual conjunto de usuários que restou não depende da ordem em que os usuários foram sendo descartados pelo algoritmo. Como em cada passo pelo menos um usuário é descartado, o mecanismo irá parar em um número linear de iterações.

Teorema 5.7. *[28, 27] Dada uma função de cost-sharing que é cross-monotonic ξ, o mecanismo $M^*(\xi)$ satisfaz as propriedades STP, PV, SC, e é group strategyproof.*

O teorema anterior fornece um novo rumo na resolução de problemas de jogos cooperativos. Agora, para um dado problema, estamos interessados em obter uma função de compartilhamento de custo que seja cross-monotonic. Uma vez encontrada tal função, todas as propriedades anteriores estarão garantidas. O esforço que faremos neste e nos próximos capítulos é justamente para tentar encontrar funções de cost-sharing que sejam cross-monotonic para os problemas.

5.3 Árvore de Steiner

Nesta seção, iremos desenvolver uma solução para o roteamento multicast que contém a propriedade budget balance e é cross-monotonic, e para qualquer conjunto D escolhido, a solução encontra uma árvore de custo no máximo duas vezes a árvore de Steiner ótima que contém a raiz e os usuários em D. A função de cost-sharing é baseada nos seguintes fatos:

- Se a função de custo nas arestas satisfaz desigualdade triangular então o custo de uma árvore geradora mínima nos vértices terminais é no máximo duas vezes o custo da solução ótima da árvore de Steiner.

- Há uma relaxação do programa linear da árvore geradora mínima que nos fornece solução inteira, ou seja, uma solução ótima.

O primeiro fato é um resultado bem conhecido na área de algoritmos de aproximação e foi apresentado no Teorema 2.10 do Capítulo 2. O segundo fato segue de um trabalho de Edmonds [6]: há uma relaxação exata para o problema da ramificação de custo mínimo (minimum branching).

Problema 5.8 (Ramificação Mínima)**.** *Dados um grafo* orientado $H = (V, \overrightarrow{E})$*, com custos* $c_e \geq 0$ *para cada aresta* $e \in \overrightarrow{E}$ *e um dos vértices* $r \in V$ *que é marcado como sendo raiz, encontrar um subgrafo conexo* $T \subseteq H$ *que gera* V *(sem perda de generalidade,* T *é uma árvore), sendo que as arestas de* T *devem ser direcionadas para a raiz* r*, ou seja, queremos uma árvore direcionada para a raiz que conecte todos os vértices. O custo da solução (que queremos minimizar) é definido como*

$$\sum_{e \in T} c_e,$$

ou seja, queremos minimizar o custo da árvore direcionada que gera V*.*

A transformação do problema da árvore geradora mínima em um grafo não-direcionado para o problema da ramificação mínima é direta. Simplesmente substituimos cada aresta $e = (u, v)$ de G por duas arestas direcionadas $(u \to v)$ e $(u \leftarrow v)$ cada qual com custo c_e, e então resolvemos o problema da ramificação mínima. Vamos denotar esse grafo direcionado por $H = (V, \overrightarrow{E})$. Uma vez que a ramificação é encontrada, ignoramos a direção das arestas para obter uma árvore geradora mínima em G.

Dizemos que um conjunto $S \subset V$ é *válido* se não é vazio e não contém a raiz. Para qualquer conjunto $S \subset V$ e $F \subset \overrightarrow{E}$ seja $\delta(S) = \{(u \to v) \in \overrightarrow{E} | u \in S \text{ e } v \in \overline{S}\}$, ou seja, informalmente falando $\delta(S)$ são as arestas que *saem* de S. A seguir, temos a formulação relaxada chamada de *relaxação do corte bidirecionado* e o dual respectivo.

Primal:

$$\begin{aligned} \min \quad & \sum_{e \in \overrightarrow{E}} c_e x_e \\ \text{s.a.} \quad & \sum_{e : e \in \delta(S)} x_e \geq 1, \qquad \forall S \text{ válido}, \\ & x_e \geq 0, \qquad \forall e \in \overrightarrow{E}. \end{aligned}$$

Dual:

$$\begin{aligned} \max \quad & \sum_{S \text{ válido}} y_S \\ \text{s.a.} \quad & \sum_{S:e\in\delta(S)} y_S \leq c_e, \qquad \forall e \in \overrightarrow{E}, \\ & y_S \geq 0, \qquad \forall S \text{ válido.} \end{aligned}$$

Dizemos que a aresta e *sente* o dual y_S se $y_S > 0$ e $e \in \delta(S)$. Assim como no Capítulo 3, dizemos que a aresta e está *justa* se a quantidade total do dual que ela sente é igual ao seu custo. O programa linear dual está tentando maximizar a soma das variáveis duais y_S sujeito à condição de que nenhuma aresta sente mais dual que o seu custo, i.e., nenhuma aresta está justa demais.

Abaixo, iremos mostrar o algoritmo de Edmonds, que é baseado na técnica primal-dual. Começando com o vetor primal $x = 0$, a solução dual trivial $y = 0$ e uma lista ordenada $F = \emptyset$, o algoritmo iterativamente "melhora a viabilidade" da solução primal e a "otimalidade" da solução dual. Quando uma aresta fica justa, ela é incluída na lista ordenada F. Seja $\delta_F(S) = \{(u \to v) \in F | u \in S \text{ e } v \in \overline{S}\}$. Em qualquer iteração, um conjunto $S \subset V$ é dito estar *violado* se é válido e $\delta_F(S) = \emptyset$. Qualquer conjunto minimal violado (CMV) é dito estar *ativo* (em relação a F). É facil ver que os conjuntos ativos devem ser disjuntos e devem ser fortemente conexos com respeito a F. De fato, quando uma aresta que liga dois conjuntos ativos se torna justa, então os dois conjuntos se tornam inativos e um novo conjunto ativo que "engloba" os dois anteriores é formado. Além disso, as arestas sempre se ajustam duas a duas, a aresta de ida e a aresta de volta, pois elas têm o mesmo comprimento.

Com o objetivo de provar propriedades deste algoritmo, iremos associar a noção de *tempo* com o algoritmo. Em cada unidade de tempo, o algorimo cresce o dual de forma unitária. Em uma dada iteração, o algoritmo encontra todos os conjuntos ativos (CMV's) e aumenta suas variáveis duais até que uma nova aresta, digamos e, fique justa. Agora, a iteração atual termina e a aresta e é adicionada à lista F. O algoritmo continua até que não exista mais conjuntos não satisfeitos e então o algoritmo "poda" F usando o procedimento da remoção reversa (i.e., remover as arestas supérfluas à ramificação direcionada em ordem reversa a que foram incluídas em F). As arestas que restam em F formam uma ramificação direcionada para a raiz.

Algoritmo 5.2: Algoritmo-Edmonds

Entrada: grafo $H = (V, \overrightarrow{E})$;
Saída: ramificação mínima F;
início
  $F \leftarrow \emptyset$;
  para cada $S \subseteq V$ **faça**
    $y_S \leftarrow 0$
  enquanto $\exists$ *CMV's com respeito a* F **faça**
    Seja $\mathcal{S}_F$ a coleção dos CMV's com respeito a F;
    Aumente simultaneamente as variáveis duais y_S de cada $S \in \mathcal{S}_F$ até alguma aresta e ficar justa;
    $F \leftarrow F \cup \{e\}$;
  Seja $F = \{e_1, e_2, \ldots, e_l\}$ a lista na ordem em que as arestas foram adicionadas;
  para $j = l$ *até* 1 **faça em paralelo** `/* Início da remoção reversa */`
    se $\nexists$ *CMV's com respeito a* $F - \{e_j\}$ **então**
      $F \leftarrow F - \{e_j\}$
  devolva F
fim

Até agora não falamos do mecanismo de cost-sharing. Relembrando, no mecanismo M^* (ver Algoritmo 5.1), estamos interessados em encontrar uma função de cost-sharing ξ que seja cross-monotonic. Como veremos adiante, esta função é definida de acordo com a solução do problema da ramificação de custo mínimo. Portanto, em cada passo do mecanismo, temos que resolver o problema da ramificação de custo mínimo para então definir a função de cost-sharing que iremos cobrar de cada usuário, para então o mecanismo decidir quais usuários ele irá descartar. Isso é feito repetidamente, até o mecanismo de cost-sharing não descartar mais nenhum usuário.

A seguir, vamos analisar a solução encontrada pelo Algoritmo 5.2.

Lema 5.9. *As arestas em F adicionadas pelo Algoritmo 5.2 formam uma solução viável para o problema da ramificação de custo mínimo.*

Demonstração. Observe que durante a construção de F, a cada passo adicionamos sempre o par de arestas de ida e volta (pois elas têm o mesmo comprimento), e no final, no passo de "poda", uma dessas arestas do par é descartada. Assim, basta mostrar que a construção de F não gera ciclos. Vamos utilizar indução para verificar isso. No caso trivial, temos dois conjuntos unitários (vértices) que ajustam o par de arestas de ida e volta, no qual uma dessas arestas é descartada no passo de poda, e formamos outro conjunto sem ciclos. Para o passo da indução, temos dois conjuntos sem ciclos, que ajustam o par de arestas de ida e volta. Note que é possível mais de um par se ajustar no mesmo instante, no entanto,

escolhemos somente um par para adicionar a F. Novamente, no passo de poda, retiramos uma das arestas deste par, ficando com um conjunto novo que não contém ciclos. ■

Lema 5.10. *Dados uma solução viável para o problema da ramificação de custo mínimo e y obtido da execução do Algoritmo 5.2, todo conjunto S válido em relação a F tal que $y_S > 0$ deve satisfazer $|\delta_F(S)| = 1$.*

Demonstração. Claramente, $|\delta_F(S)| = 0$ não caracteriza uma solução válida. Vamos supor, por contradição, que para algum S, $|\delta_F(S)| > 1$. Então há pelo menos duas arestas que "saem" de S, vamos chamá-las de $(u \to v)$ e $(u' \to v')$, onde $u, u' \in S$ e $v, v' \notin S$. Como a solução é válida, v e v' vão se encontrar em algum ponto da ramificação direcionada, digamos w. Como $u, u' \in S$, então há um caminho direcionado de u para u' ou então de u' para u. Vamos supor, sem perda de generalidade, que existe um caminho de u para u'. Então u pode chegar à raiz por dois caminhos: pelo caminho que passa por u, v, w e então a raiz ou pelo caminho que passa por u, u', v', w e então até a raiz. Isto contraria a hipótese da solução ser uma ramificação direcionada viável. ■

Teorema 5.11. *O custo da ramificação obtida é precisamente igual à soma das variáveis duais que foram crescidas, i.e.,*

$$\sum_{e \in F} c_e = \sum_{S \text{ válido}} y_S.$$

Demonstração. O crescimento dual y_S dos conjuntos válidos contribuem pelo pagamento total das arestas da solução. No entanto, um único conjunto S poderia estar contribuindo para várias arestas, fazendo com que $\sum_{e \in F} c_e \geq \sum_{S \text{ válido}} y_S$. No entanto, pelo Lema 5.10, sabemos que cada conjunto contribui para somente uma aresta. Como as arestas são pagas totalmente pelo crescimento dual, então o teorema segue. ■

Esse importante teorema nos mostra que a ramificação encontrada pelo algoritmo é ótima, pois pelo Teorema Fraco da Dualidade, quando uma solução viável primal tem o mesmo valor que uma solução viável dual, então essas duas soluções são ótimas. Usando este fato, poderemos mostrar que nossa função de cost-sharing é budget-balanced, como veremos adiante.

Iremos agora definir a função de cost-sharing ξ. O custo-compartilhado para um usuário $i \in D$ é calculado da seguinte forma. Seja T o primeiro instante de tempo em que há um caminho de arestas justas de i para a raiz. Seja $S(t)$ o conjunto de vértices que i alcança usando arestas justas no instante de tempo $t \leq T$. Em cada instante de tempo $t < T$, o algoritmo cresce a variável dual $y_{S(t)}$ à razão unitária. A função de cost-sharing para o usuário $i \in D$ é dada por

$$\xi_{\text{AGM}}(D, i) = \int_0^T \frac{1}{|S(t)|} dt.$$

Intuitivamente falando, no instante de tempo t, a função de cost-sharing cobra de cada usuário em $S(t)$ o custo proporcional ao número de usuários que participam de $S(t)$, ou seja, o crescimento dual é dividido igualmente entre todos os usuários do conjunto.

Teorema 5.12. *A função de cost-sharing ξ_{AGM} é budget-balanced.*

Demonstração. Observe que a cada instante de tempo, a função de cost-sharing está simplesmente distribuindo o dual crescente $y_{S(t)}$ entre os usuários em $S(t)$. Assim,

$$\sum_{i \in D} \xi_{\text{AGM}}(D, i) = \sum_{S \text{ válido}} y_S = \text{val}(O, D),$$

onde O é a solução encontrada e a última igualdade vem do fato que o dual total construído é igual ao custo da árvore encontrada (solução primal). ■

Teorema 5.13. *A função de cost-sharing ξ_{AGM} é cross-monotonic.*

Demonstração. Considere uma execução do Algoritmo 5.2 em um conjunto de usuários D e seja $w \notin D$ um outro usuário. Seja $i \in D$ um usuário qualquer. Sejam R e R' as execuções do algoritmo na entrada D e $D \cup \{w\}$ respectivamente. Sejam $S(t)$ e $S'(t)$ os conjuntos de vértices que i alcança através de arestas justas no instante de tempo t nas execuções R e R' respectivamente. Sejam T e T' os primeiros instantes de tempo em que há um caminho de arestas justas de i até a raiz nas execuções R e R' respectivamente. Definimos,

$$\xi_{\text{AGM}}(D, i) = \int_0^T \frac{1}{|S(t)|} dt,$$

e

$$\xi_{\text{AGM}}(D \cup \{w\}, i) = \int_0^{T'} \frac{1}{|S'(t)|} dt.$$

Se num dado instante de tempo t, i pode alcançar w na execução R', então $S'(t) \supset S(t)$, e caso contrário $S'(t) = S(t)$. Além disso, i alcança a raiz no mesmo instante ou mais cedo na execução R' que na execução R, ou seja, $T' \leq T$. Portanto $\xi(D \cup \{w\}, i) \leq \xi(D, i)$. ■

Corolário 5.14. *Utilizando o mecanismo M^* com a função de cost-sharing ξ definida para o problema da árvore geradora mínima, garantimos as propriedades STP, PV, SC e group strategyproof.*

Apesar desta seção prometer resultados para o problema da árvore de Steiner de custo mínimo, devemos antes introduzir alguns conceitos de aproximalidade na condição *Budget-Balance*, e desta forma, utilizar o resultado da árvore geradora mínima para obter uma função de cost-sharing "aproximada" para o problema da árvore de Steiner de custo mínimo. Os detalhes disto serão vistos na Seção 5.4.

5.4 Relaxando a condição de *Budget-Balance*

Obviamente, a função de cost-sharing deve ser tal que o provedor de serviço não tenha prejuízo. Na presença de competição, não deve-se criar um grande excesso de lucro também. Budget-balance garante ambas estas condições, contudo esta é uma propriedade difícil de ser garantida. Além disso, mesmo se satisfeita, pode sofrer da seguinte falha. Vamos considerar nossa solução para o problema da árvore de Steiner. Nós assumimos que estávamos trabalhando com um grafo completo G, com as arestas satisfazendo a desigualdade triangular, e encontramos a árvore geradora de custo mínimo T neste grafo. No entanto, a rede original, H, pode não ter todos os enlaces que conectam todos os pares de nós. O grafo G é obtido pelo fechamento de H, tal que as arestas em G correspondem aos caminhos mínimos de H. Ao mapear a árvore T novamente para H, substituímos arestas por caminhos. Isso nos dá um grafo gerador, digamos H', que provavelmente contém ciclos e arestas múltiplas. Agora, a propriedade budget-balance requer que a companhia provedora de serviços envie uma mensagem em cada link de H', que é claramente um desperdício. Qualquer árvore geradora que é um subgrafo de H', digamos T', é suficiente.

Esse tipo de falha e a dificuldade em garantir a propriedade budget-balance nos motiva à seguinte definição. Seja $\text{val}(O^*, D)$ o custo da solução ótima O^* para servir os usuários em D e $\alpha \geq 1$ uma função de n (geralmente uma constante). Dizemos que a função de cost-sharing ξ é *α-aproximada* se satisfaz:

1. **Competitividade α-aproximada** $\forall D \subseteq V : \sum_{i \in D} \xi(D, i) \leq \alpha \cdot \text{val}(O^*, D)$, i.e., qualquer conjunto D de usuários pagam no máximo α vezes o custo ótimo de servir D.

2. **Recuperação do custo** $\forall D \subseteq V : \sum_{i \in D} \xi(D, i) \geq \text{val}(O, D)$, para alguma solução O obtida para o problema de custo $\text{val}(O, D)$ (não necessariamente a solução ótima), i.e., o custo que o servidor paga deve ser recuperado pelos usuários.

3. se $i \notin D$ então $\xi(D, i) = 0$.

Escrevendo de uma maneira análoga, temos que

$$\text{val}(O, D) \leq \sum_{i \in D} \xi(D, i) \leq \alpha \cdot \text{val}(O^*, D).$$

Uma maneira alternativa de chamar uma função de compartilhamento de custo que obedece às inequações acima é *função de cost-sharing com competitividade α-aproximada.*

Na verdade, a escolha de relaxar a competitividade é arbitrária, poderíamos também relaxar a recuperação do custo. A única diferença é que a função de cost-sharing teria de

ser multiplicada pelo fator $1/\alpha$. Desta forma, na literatura, também encontramos o termo *função de cost-sharing com recuperação do custo α-aproximada*, que é definido como

$$\frac{1}{\alpha} \cdot \text{val}(O, D) \leq \sum_{i \in D} \xi'(D, i) \leq \text{val}(O^*, D),$$

onde $\xi'(D, i) = (1/\alpha) \cdot \xi(D, i)$.

Com algumas modificações no teorema de Moulin e Shenker, chega-se num teorema equivalente que usa a definição de função de cost-sharing que é α-aproximado.

Teorema 5.15. *[28, 27] Dada uma função de cost-sharing que é cross-monotonic α-aproximado ξ, o mecanismo $M^*(\xi)$ satisfaz as propriedades STP, PV, SC, e é group strategyproof.*

Com base nas definições de budget-balance aproximado, no Corolário 5.14, no Teorema 2.10 e no Teorema 5.15, a árvore T' é uma solução com competitividade 2-aproximada, como enuncia o seguinte teorema.

Teorema 5.16. *Há um mecanismo de cost-sharing com competitividade 2-aproximado, STP, PV, SC e group strategyproof para o jogo da árvore de Steiner mínima.*

Outro bom exemplo das consequências da definição de budget-balance aproximado, que iremos só comentar, é o *jogo do caixeiro viajante métrico*, no qual o custo das arestas satisfaz desigualdade triangular e a viagem do caixeiro começa no vértice raiz e passa por todos outros vértices usuários que foram escolhidos pelo mecanismo de cost-sharing. Sem aprofundarmos em detalhes, um algoritmo conhecido [35] duplica as arestas de uma árvore geradora mínima, encontra um circuito Euleriano e executa um passo de atalho (short cut) nos fornecendo um algoritmo com fator de aproximação 2. Usando este fato e o Teorema 5.15, temos o seguinte.

Teorema 5.17. *Há um mecanismo de cost-sharing com competitividade 2-aproximado, STP, PV, SC e group strategyproof para o jogo do caixeiro viajante métrico.*

Capítulo 6

Técnica primal-dual

Neste capítulo, iremos aplicar a técnica primal-dual vista no Capítulo 3 para obter funções de compartilhamento de custo. Os métodos descritos neste capítulo nos fornecem um esquema genérico para compartilhar custos utilizando a técnica primal-dual. Para exemplificar o método, chamado de *processo fantasma*, iremos começar com o problema Facility Location. Depois, resolveremos os problemas Rent-or-Buy e Connected Facility Location.

Este capítulo está baseado nos trabalhos de Pal e Tardos [31] e Leonardi e Schaefer [22]. Iremos apresentar uma função de cost-sharing cross-monotonic que é budget-balance 3-aproximado para o problema Facility Location, baseado no trabalho de Pal e Tardos. No trabalho de Immorlica, Mahdian e Mirrokni [15], foi mostrado que não existe função de cost-sharing cross-monotonic que melhora este resultado, mas omitiremos este resultado neste documento. Ainda no trabalho de Pal e Tardos, foi obtida uma função de cost-sharing cross-monotonic que é budget-balance 15-aproximado para o problema Rent-or-Buy, utilizando a mesma idéia de resolução do problema Facility Location. No trabalho de Leonardi e Schaefer, foi obtida uma função de cost-sharing cross-monotonic que é budget-balance 30-aproximado para o problema Connected Facility Location. Os resultados de ambos trabalhos serão apresentados neste capítulo.

6.1 Facility Location

Nesta seção, iremos obter uma função de cost-sharing que é cross-monotonic para o problema Facility Location métrico (ver Problema 2.3). Para o problema Facility Location, existem algoritmos primais-duais elegantes e simples, e portanto podemos utilizar esse problema com um exemplo "limpo" para ilustrar a técnica que iremos utilizar neste capítulo, chamada de *processo fantasma*, para depois aplicarmos em problemas de projeto de redes mais complexos, resolvendo então o Rent-or-Buy e o Connected Facility Location.

Uma maneira natural de obter uma função de cost-sharing é considerar a técnica

primal-dual, explicada no Capítulo 3. A técnica primal-dual é baseada numa maneira "justa" de dividir o custo da solução: todos usuários aumentam seus custos de contribuição uniformemente até todos estarem satisfeitos. No final, o custo compartilhado de um usuário é o quanto este contribuiu para montar a solução do problema. Este fato pode ser utilizado como uma tentativa para se obter uma função de cost-sharing para o problema. No entanto, nem sempre uma função definida dessa maneira tem a propriedade de ser cross-monotonic. Como exemplo, podemos pensar no algoritmo primal-dual de Jain e Vazirani [18] para o problema Facility Location, apresentado no Algoritmo 2.1 do Capítulo 2. Ao adicionar usuários bem próximos a um usuário i, faz com que i esteja satisfeito mais rapidamente (pois paga a facilidade mais rapidamente). No entanto, como i pára de contribuir mais cedo, ele pode contribuir com menos para as outras facilidades, fazendo com que os outros usuários que já estavam no problema tenham que pagar mais para ficar satisfeitos. Podemos ver isto no Exemplo 6.1.

Exemplo 6.1. Considere uma instância com duas facilidades p e q tal que $f_p = 2$ e $f_q = 1$, e dois clientes j_1 e j_2. Sejam as distâncias $c_{j_1,p} = c_{j_2,p} = c_{j_2,q} = 1$ (veja Figuras 6.1 e 6.2), as outras distâncias são definidas pela função de caminho mínimo. O algoritmo de Jain e Vazirani irá aumentar ambos α_{j_1} e α_{j_2} até a facilidade p ficar paga, e então cada cliente é declarado satisfeito e seus α's param de crescer. Portanto, cada cliente termina pagando pela sua distância à p, e o custo de p é dividido uniformemente entre eles. Os custos compartilhados resultantes são $\xi(\{j_1, j_2\}, j_1) = 2$ e $\xi(\{j_1, j_2\}, j_2) = 2$, como mostra a Figura 6.1.

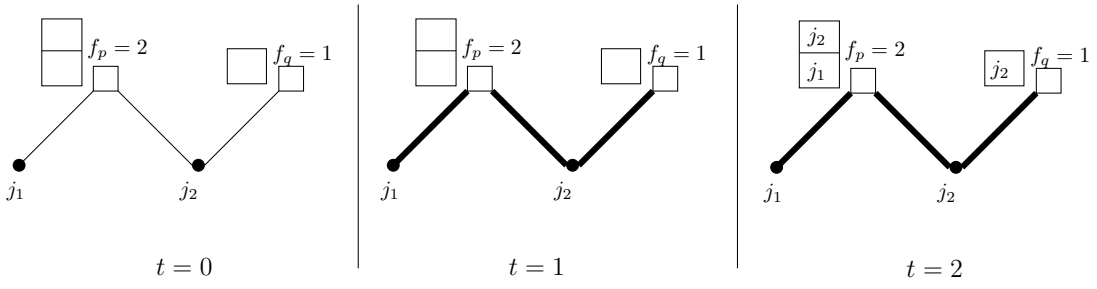

Figura 6.1: O algoritmo de Jain e Vazirani sobre j_1 e j_2. Os quadrados ao lado das facilidades denotam por qual cliente foi pago o custo da facilidade.

Agora vamos supor que adicionamos uma novo cliente j_3 localizado muito próximo à facilidade q, i.e., $c_{j_3,q} = 0$. A Figura 6.2 ilustra esta situação. Agora, temos uma nova situação: o usuário j_3 começa a contribuir imediatamente, fazendo com que q seja pago no instante de tempo 1. Assim, o usuário j_2 ficará satisfeito no instante de tempo 1 quando se conectar com q, parando de crescer seu α_{j_2}. Isso significa que j_2 não contribui para p,

e portanto j_1 tem que pagar por p sozinho. Assim, $\xi(\{j_1, j_2, j_3\}, j_1) = 3 > \xi(\{j_1, j_2\}, j_1)$, e a função de cost-sharing não é cross-monotonic.

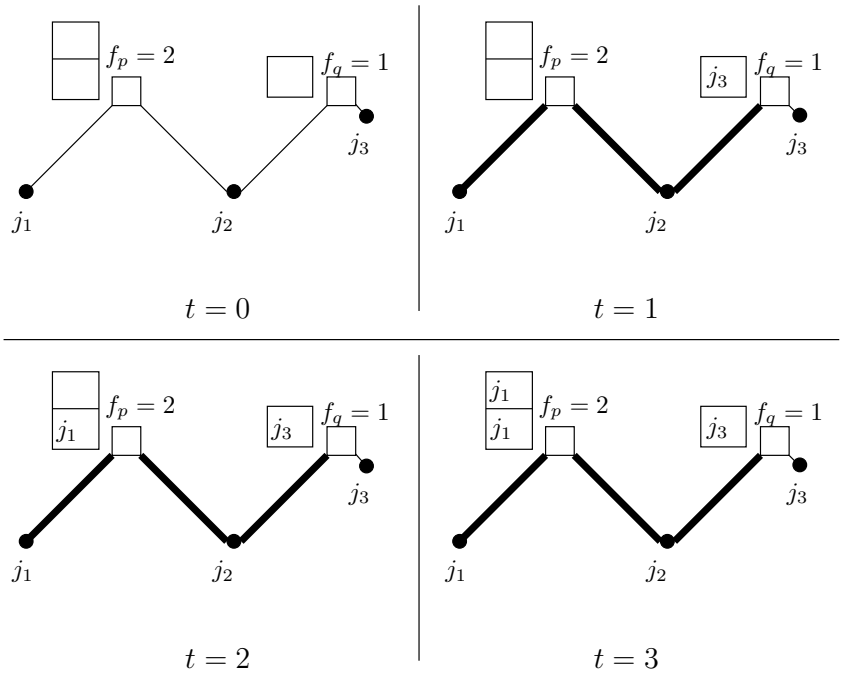

Figura 6.2: O algoritmo de Jain e Vazirani sobre j_1, j_2 e j_3. Os quadrados ao lado das facilidades denotam por qual cliente foi pago o custo da facilidade.

A idéia que nos permitirá obter funções de cost-sharing cross-monotonic é ter uma função de cost-sharing "fantasma" para cada usuário. Embora no Algoritmo 2.1 nós sejamos forçados a parar de crescer a contribuição de um usuário para não sobrecarregar algum grupo de usuários, nós continuamos crescendo a função fantasma até que todos os usuários fiquem satisfeitos. Em outras palavras, teremos duas funções de custo, uma "real", e outra "fantasma", e o principal desafio é conseguir comparar essas duas funções para que, mesmo que a maioria das facilidades esteja somente paga "virtualmente" pelos custos fantasmas, o custo-distribuído real ainda pague suficiente para construir uma solução viável. Por exemplo, no problema Facility Location, a contribuição adicional dos fantasmas implica em abrir mais facilidades, e conseqüentemente os usuários se conectarem antes. Esta idéia garante trivialmente uma função cross-monotonic, pois quanto mais usuários (e seus fantasmas) adicionarmos, mais facilidades serão abertas e cada usuário ficará satisfeito mais cedo.

6.1.1 O processo fantasma

Para definir a função de cost-share, usaremos o "processo fantasma", que pode ser visto como uma versão suavizada (que evita as paradas súbitas) de um algoritmo de aproximação primal-dual, como por exemplo, o Algoritmo 2.1 de Jain e Vazirani. O *processo fantasma* cresce um fastasma para cada usuário. O fantasma do usuário j é uma bola (raio) com j no centro, que cresce uniformemente para o infinito. O raio do fantasma em algum instante de tempo $t > 0$ é igual a t. O fantasma de um usuário j *toca* a facilidade p no instante de tempo t se $t \geq c_{j,p}$. Inicialmente, as facilidades são consideradas *não-cheias*. Todos fantasmas que tocam uma facilidade não-cheia começam a contribuir para pagar a facilidade. A contribuição de um fantasma j para a facilidade p no instante de tempo t é dada por $\kappa_{j,p} = \max\{0,\ t - c_{j,p}\}$. Uma facilidade p é declarada *cheia* se $\sum_{j \in D} \kappa_{j,p} \geq f_p$. Seja $t(p)$ o instante de tempo em que p fica cheia. Seja S_p o conjunto de usuários cujos fantasmas contribuem para o pagamento de p, i.e., todos usuários que estão numa distância estritamente menor que $t(p)$ de p. Note que se cobrarmos de cada usuário em S_p a quantidade $\alpha = t(p)$ iremos recuperar exatamente o custo de abrir a facilidade e o custo de conectar todos os usuários em S_p a p, pois $f_p = \sum_{j \in S_p}(t(p) - c_{j,p})$ por definição.

Algoritmo 6.1: Fantasma-FacilityLocation

Entrada: grafo $G = (V, E)$, demandas D, facilidades F;
Saída: $t(p), S(p)$ para toda facilidade p cheia;
início
  $t \leftarrow 0$;
  enquanto $\exists j \in D$ *não conectado a alguma facilidade cheia* **faça**
    aumente t (fazendo $\kappa_{j,p} \leftarrow \max\{0,\ t - c_{j,p}\},\ \forall j \in D, p \in F$) até que:
    (a) $\sum_{j \in D} \kappa_{j,p} \geq f_p$, ou
    (b) $\exists j \in D$ e facilidade p cheia, tal que $t \geq c(j, p)$
    se *(a) foi satisfeita para algum* $p \in F$ **então**
      p é declarada cheia;
      $t(p) \leftarrow t$;
      $S(p) \leftarrow \{j \in D | c(j, p) < t(p)\}$;
    se *(b) foi satisfeita para algum* $j \in D$ *e* $p \in F$ **então**
      j é conectada a p;
  devolva $t(p), S(p)$ *para toda facilidade* p *cheia*
fim

6.1.2 Função de cost-sharing

A cada iteração, aumentamos a variável α_j de cada usuário j a razão unitária até o fantasma de j tocar alguma facilidade cheia ou alguma facilidade que j está tocando se tornar cheia. Mais formalmente,

$$\alpha_j = \min\left(\min_{p:j\in S_p} t(p), \quad \min_{p:j\notin S_p} c_{j,p}\right).$$

A Figura 6.3 ilustra a diferença entre a variável α acima definida e a variável α definida pelo Algoritmo 2.1 de Jain e Vazirani. Observe que o processo fantasma é utilizado para definir a função de cost-sharing. Assim,

$$\xi(D, j) = \alpha_j.$$

Nas seções seguintes também usaremos o processo fantasma com este mesmo objetivo, no entanto, usaremos variações dos valores de α_j para definir a função final de cost-sharing.

Note que ao adicionar mais usuários, cada facilidade pode somente ser paga mais rapidamente, e portanto cada usuário irá se tornar satisfeito mais cedo. Partindo desta idéia, podemos provar o Teorema 6.2.

Antes disso, vamos fazer uma observação quanto a notação. Sabemos que $\xi(D, j) = \alpha_j$ para todo $j \in D$. Assim, durante algumas provas desta seção, eventualmente usaremos α_j para denotar o valor do custo compartilhado, por acharmos essa notação mais intuitiva com relação às idéias de variáveis duais e bolas fantasmas.

Teorema 6.2. *A função de cost-sharing ξ é cross-monotonic.*

Demonstração. Vamos supor que adicionamos um usuário u ao sistema. Seja α o valor do custo compartilhado definido para o conjunto de usuários D e α' o valor do custo compartilhado definido para o grupo de usuário $D \cup \{u\}$. Analogamente, definimos $t(p), t'(p)$ como os instantes de tempo em que uma facilidade p fica cheia nas execuções sobre D e $D \cup \{u\}$ respectivamente.

Devemos mostrar que $\xi(D \cup \{u\}, j) \leq \xi(D, j)$ para todo $j \in D$. Portanto, é suficiente mostrar que $\alpha' \leq \alpha$. Note que, ao incluir um novo usuário, o fantasma deste com certeza vai contribuir para que uma facilidade seja paga mais rapidamente, em outras palavras, para qualquer facilidade p, $t'(p) \leq t(p)$, portanto $\min_{p:j\in S_p} t'(p) \leq \min_{p:j\in S_p} t(p)$. Pode acontecer na execução sobre D que um dado $j \in S_p$ para algum p, mas na execução sobre $D \cup \{u\}$, $j \notin S_p$ e então $c_{j,p}$ será levado em consideração na função mínimo de α' (este caso pode acontecer pelo fato da execução sobre $D \cup \{u\}$ terminar antes). Mas neste caso $c_{j,p} \leq t(p)$. Para os outros casos, em que $j \notin S_p$ tanto na execução sobre D como sobre $D \cup \{u\}$, os $c_{j,p}$ permanecem inalterados. Portanto $\alpha' \leq \alpha$. ■

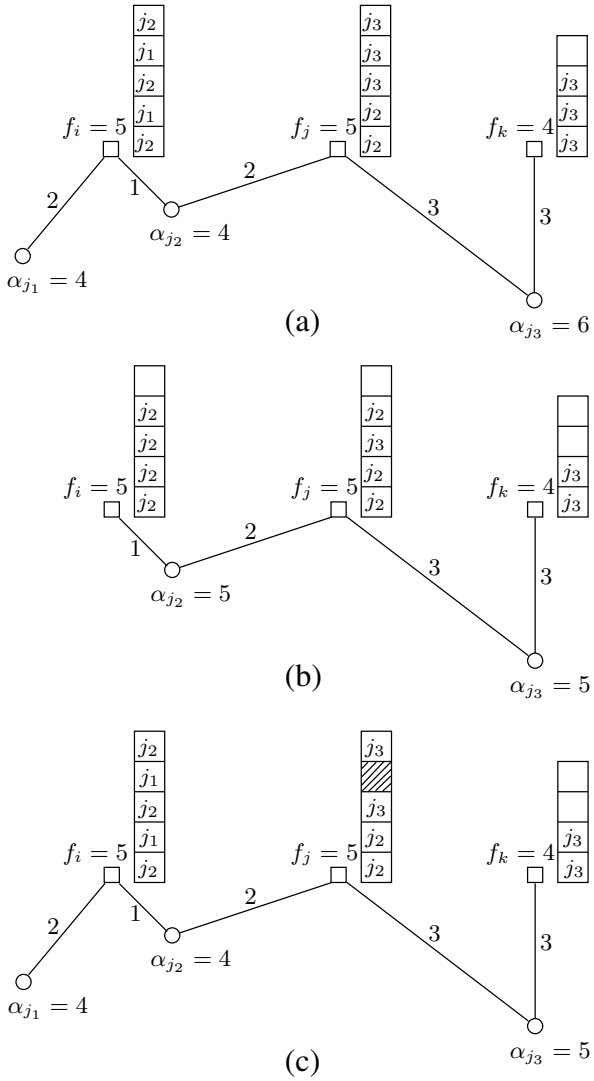

Figura 6.3: Os quadrados ao lado das facilidades denotam por qual cliente foi pago o custo da facilidade. Nas figuras (a) e (b), estamos utilizando o algoritmo de Jain e Vazirani. Note que na ilustração (a), $\alpha_{j_3} = 6$ e na ilustração (b), $\alpha_{j_3} = 5$, demonstrando que o algoritmo não é cross-monotonic. A figura (c) é o caso onde estamos considerando os cost-shares definidos acima. O quadrado hachurado representa o que foi pago pelo crescimento fantasma. Note que o algoritmo executa até o instante de tempo $t = 5$, e neste instante ambos j_2 e j_3 pagam com seus fantasmas a facilidade j. Mas pela definição do valor final do custo compartilhado, $\alpha_{j_2} = 4$ e $\alpha_{j_3} = 5$.

6.1.3 O processo real: decidindo quais facilidades abrir

Para cada facilidade p, decidimos se esta será ou não será aberta no momento em que p fica cheia. Abrimos a facilidade p se e somente se no instante de tempo $t(p)$ não há

alguma facilidade q já aberta tal que $c_{p,q} \leq 2t(p)$, onde $c_{p,q}$ é o custo do caminho mínimo de p a q. Isto irá manter as facilidades abertas suficientemente longe umas das outras.

6.1.4 Análise

Teorema 6.3 (Competitividade)**.** *Para toda solução O de custo val(O, D) do Facility Location (por exemplo, uma solução ótima) vale $\sum_{j \in D} \alpha_j \leq$ val(O, D).*

Demonstração. Podemos assumir sem perda de generalidade que uma solução ótima consiste de uma única facilidade aberta p, pois o caso geral pode ser obtido considerando a soma sobre todos as facilidades abertas. Mas sabemos que os cost-shares dos usuários em S_p não ultrapassam o custo de p mais os custos de conectar eles a p, ou seja, $\sum_{j \in D: j \in S_p} \alpha_j \leq f_p + \sum_{j \in D: j \in S_p} c_{j,p}$. Além disso, os cost-shares de cada usuário fora de S_p não ultrapassam o custo de conectar eles a p, ou seja, $\sum_{j \in D: j \notin S_p} \alpha_j \leq \sum_{j \in D: j \notin S_p} c_{j,p}$. Combinando os dois temos

$$\begin{aligned}
\sum_{j \in D} \alpha_j &= \sum_{j \in D: j \in S_p} \alpha_j + \sum_{j \in D: j \notin S_p} \alpha_j \\
&\leq f_p + \sum_{j \in D: j \in S_p} c_{j,p} + \sum_{j \in D: j \notin S_p} c_{j,p} \\
&= f_p + \sum_{j \in D} c_{j,p} \\
&= \text{val}(O, D).
\end{aligned}$$

■

A seguir iremos relacionar os custos da função de cost-sharing ao custo do processo real, que abre um subgrupo de facilidades.

Proposição 6.4. *Se as facilidades p e q estão abertas, então seus conjuntos de contribuintes S_p e S_q são disjuntos.*

Demonstração. Podemos assumir sem perda de generalidade que $t(p) \geq t(q)$, pois caso contrário, a prova é análoga. Seja $c_{p,q}$ o custo do caminho mínimo entre p e q. Como nós decidimos abrir p, ele deve estar suficientemente longe de q: $c_{p,q} \geq 2t(p) \geq t(p) + t(q)$. Vamos supor por contradição que existe um usuário j que contribui para ambos p e q. Então $\alpha_j \leq t(q)$, pois $t(q) \leq t(p)$. Assim, usando a desigualdade triangular, temos

$$\begin{aligned}
t(p) + t(q) &\leq c_{p,q} \\
&\leq c_{j,p} + c_{j,q} \\
&\leq 2\alpha_j \\
&\leq 2t(q),
\end{aligned}$$

Mas note que, pelo fato que $t(p) \geq t(q)$, a desigualdade obtida $t(p) + t(q) \leq 2t(q)$ não é válida quando esta é estritamente menor, gerando uma contradição com o fato de j contribuir aos dois conjuntos. No entanto, a desigualdade obtida é válida na igualdade quando $t(p) = t(q)$. Neste caso, $\alpha_j = t(q) = t(p)$, mas j não pode contribuir para ambos p e q pois $c_{p,q} = 2t(q)$ e na melhor das hipóteses j consegue encostar seu raio em ambos p e q, mas não contribui com nada excedente para ambos. ■

O lema anterior significa que todo usuário contribui no pagamento de no máximo uma facilidade. Para propósitos de análise, iremos atribuir todos usuários do conjunto S_p à facilidade p, para cada facilidade aberta p. Atribuímos cada usuário que não pertence a nenhum conjunto S_p de alguma facilidade aberta p a sua facilidade aberta mais próxima.

Sabemos pelo Teorema 6.3 que a soma de todos α's atribuídos é sempre menor que uma solução de custo ótimo, mas queremos garantir que esta soma seja pelo menos 1/3 da solução obtida (não confundir com a solução ótima), ou seja, queremos

$$\frac{\text{val}(O, D)}{3} \leq \sum_{j \in D} \alpha_j \leq \text{val}(O^*, D),$$

onde $\text{val}(O, D)$ é a solução O com respeito à atribuição descrita no parágrafo anterior e $\text{val}(O^*, D)$ é o custo de uma solução ótima O^*.

Vamos afirmar que os usuários no conjunto S_p de cada facilidade aberta p pagam o suficiente para cobrir 1/3 do custo de p, assim como pagam 1/3 do custo de conexão. Pela definição de S_p, $f_p = \sum_{j \in S_p}(t(p) - c_{j,p})$. Manipulando, obtemos que o custo de abrir f_p mais o custo de conexão $\sum_{j \in S_p} c_{j,p}$ é igual a $|S_p|t(p)$. Para provar nossa afirmação é suficiente mostrar que o custo compartilhado de cada usuário é pelo menos $t(p)/3$.

Lema 6.5. *Seja p uma facilidade aberta, e $j \in S_p$ um usuário. Então $3\alpha_j \geq t(p)$.*

Demonstração. Considere um usuário $j \in S_p$. Por contradição, assuma que $\alpha_j < t(p)/3$. Seja q a primeira facilidade cheia que j tocou,i.e.,$c_{j,q} \leq \alpha_j$ e $t(q) \leq \alpha_j$. Se q está aberta, temos $c_{p,q} \leq c_{j,p} + c_{j,q} \leq t(p) + \alpha_j < 2t(p)$, que é uma contradição pois q e p não podem estar abertas tão próximas, como decidido anteriormente. Se q não está aberta, então existe uma facilidade aberta q' tal que $c_{q,q'} \leq 2t(q) \leq 2\alpha_j$. Veja a Figura 6.4.

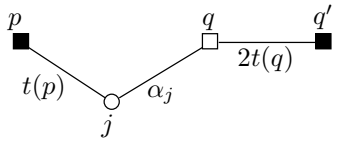

Figura 6.4: Se $\alpha_j < t(p)/3$, então $c_{p,q'} < 2t(p)$.

Assim temos

$$\begin{aligned} c_{p,q'} &\leq c_{p,j} + c_{j,q} + c_{q,q'} \\ &\leq t(p) + \alpha_j + 2t(q) \\ &\leq t(p) + 3\alpha_j \\ &< 2t(p), \end{aligned}$$

onde a última desigualdade vem do fato que assumimos $\alpha_j < t(p)/3$, e novamente temos uma contradição. Note que $p \neq q'$, pois $t(q') \leq \alpha_j < t(p)$. ■

O Lema 6.5 considera os usuários $j \in S_p$, para alguma facilidade aberta p, mostrando que estes pagam por 1/3 do custo de conexão e de abertura da facilidade. De maneira análoga à prova deste lema, podemos mostrar que os custos compartilhados dos usuários que não pertencem a nenhum S_p de uma facilidade aberta p pagam por 1/3 dos seus custos de conexão (não é necessário mostrar que pagam pela abertura de facilidades).

Lema 6.6. *Para cada usuário j que não pertence a nenhum S_p, há uma facilidade aberta p tal que $3\alpha_j \geq c_{j,p}$.*

Demonstração. Seja q a localidade que define α_j, i.e., $\alpha_j \geq t(q)$ e $\alpha_j \geq c_{j,q}$. Como q não é aberta no instante de tempo $t(q)$, deve existir uma localidade p aberta tal que $c_{q,p} \leq 2t(q)$. Portanto, pela desigualdade triangular $c_{j,p} \leq \alpha_j + 2t(q) \leq 3\alpha_j$. ■

Teorema 6.7 (Recuperação do custo)**.** *O custo da solução construída é no máximo $3\sum_{j \in D} \alpha_j$.*

Demonstração. Os Lemas 6.5 e 6.6 dizem que todos os usuários recuperam 1/3 do custo, portanto o teorema segue. ■

Assim, como conclusão dos teoremas de competitividade e recuperação de custo, temos o seguinte teorema.

Teorema 6.8. *Há uma função de cost-sharing que é budget-balance 3-aproximado e group strategyproof para o jogo do Facility Location.*

6.2 Rent-or-Buy

Como vimos no Capítulo 3, uma solução aproximada para este problema pode ser construída em duas fases: primeiro usamos um algoritmo de Facility Location para obter agrupamentos das demandas e depois construímos uma árvore de Steiner no conjunto das demandas agrupadas. Seria natural tentar combinar as funções de custo-compartilhado

cross-monotonic para o Facility Location (vista na seção anterior) e para a árvore de Steiner (vista no Capítulo 5). No entanto, combinar uma função de cost-sharing para um processo de duas fases não nos fornece uma função de cost-sharing cross-monotonic: as decisões feitas no passo de agrupamento afetam de maneira imprevisível o modo como os jogadores contribuem na construção da árvore, sendo impossível garantir que a função seja cross-monotonic na segunda fase.

Assim, para superar a dificuldade de criar um processo de duas-fases que seja cross-monotonic, executamos os dois processos simultaneamente, adicionando todos os pontos de agrupamento em potencial na segunda fase tão logo eles são criados na primeira fase. Deste modo temos a seguinte idéia para a função cross-monotonic: quanto mais usuários temos, mais pontos de agrupamentos ficam abertos e mais cedo. Mais pontos de agrupamento significa que também a árvore é construída mais rapidamente, somente reduzindo o cost-share de cada usuário.

Assim como no caso do Facility Location, o principal desafio é provar que os custos compartilhados reais podem pagar por uma solução viável. Note que, como a primeira fase pode abrir pontos de agrupamento além do necessário (de fato, cada ponto alcançado por suficientes usuários se torna um ponto de agrupamento), o custo de construir a árvore de Steiner em todos eles pode ser muito alto. Portanto precisamos selecionar somente um subgrupo desses pontos, e comparar o processo que constrói a árvore nos pontos selecionados com o processo que considera todos os pontos em potencial, para mostrar que a função de cost-sharing obtida pelo segundo processo pode pagar pela árvore construída pelo primeiro processo.

Novamente, iremos crescer bolas ao redor de cada usuário. Como não há custo de abertura, iremos abrir um *centro* em alguma localidade p que foi tocada por M ou mais bolas. Um algoritmo primal-dual comum, como por exemplo o algoritmo mostrado no Capítulo 3, iria congelar todos os usuários que se conectassem a algum centro aberto e pararia de crescer suas bolas. Mas para garantir que a função de custo-compartilhado seja cross-monotonic, usaremos o mesmo truque que utilizamos com o Facility Location: continuamos crescendo a bola fantasma ao redor de todo usuário, mesmo após um usuário ficar conectado. Note que, como as bolas fantasmas continuam contribuindo para abrir novos centros, muitos mais centros são abertos do que o necessário, dando origem a um novo problema: cada usuário pode estar conectado a múltiplos centros, e o custo da árvore de Steiner a ser construída sobre todos os centros é muito grande.

Vamos tratar este problema de duas maneiras diferentes. Quando estamos calculando os custos compartilhados, vamos assumir que estamos construindo uma árvore de Steiner em todos os centros. Então dividimos o custo de cada centro e tentamos cobrar igualmente de todos os usuários conectados ao centro. Agora, como um usuário pode estar conectado a múltiplos centros, ele deve escolher com qual centro deve contribuir (vamos assumir

que cada usuário quer contribuir somente com o centro onde a contribuição requerida é a menor).

Como a maioria dos centros não tem contribuição de suficientes usuários (ou talvez de nenhum usuário), não podemos esperar que a contribuição recupere uma fração constante de custo de uma árvore de Steiner em todos os centros. Para construir uma árvore que a função de cost-sharing possa pagar, selecionamos somente um subgrupo dos centros, e cada usuário é atribuído a somente um centro. Agora, forçamos um usuário a contribuir somente com o centro a que está atribuído, mas ainda assim só podemos obter a contribuição que é igual ao mínimo das contribuições em potencial para qualquer centro possível.

Tecnicamente, a parte difícil é relacionar os dois processos: um que assume construir a árvore de Steiner em todos os centros, usado para definir a função de cost-sharing, e o processo primal-dual padrão que usamos para construir a árvore de Steiner real em um subconjunto de centros, e mostrar que os cost-shares do primeiro processo pagam pela solução construída pelo segundo processo.

Iremos novamente utilizar neste problema as Suposições 3.1 (demandas unitárias), 3.2 (raiz na solução) e 3.3 (métrica infinitesimal nas arestas), descritas no Capítulo 3.

O principal resultado desta seção é obter uma função de cost-sharing para o Rent-or-Buy, como enuncia o seguinte teorema.

Teorema 6.9. *Há um método de cost-sharing cross-monotonic para o jogo rent-or-buy que é competitivo e recupera pelo menos* $1/15$ *do custo de uma rede viável que pode ser construída em tempo polinomial.*

6.2.1 O processo fantasma

O processo fantasma considerado nesta seção é parecido com o processo utilizado para o Facility Location. Portanto, cada usuário j tem um fantasma que é uma bola $B(j,t)$ com j no centro e raio igual ao tempo t. A diferença é que, ao invés de pensar que os fantasmas contribuem para abrir uma facilidade, iremos estar procurando por localidades que são potenciais pontos de agrupamento. Bons candidatos são localidades que foram tocadas por M ou mais fantasmas. Vamos denotar o conjunto (que varia com o tempo) de todas essas localidades por $\mathcal{C}$. Note que em qualquer instante de tempo, o conjunto $\mathcal{C}$ pode ser representado como uma coleção de vértices, arestas e segmentos de arestas.

Um *componente conexo* de $\mathcal{C}$ é qualquer subconjunto C de $\mathcal{C}$ maximal na inclusão tal que quaisquer duas localidades $p_1, p_2 \in C$ são ligados por um caminho (uma curva contínua) que também está em C.

Vamos observar mais de perto como o conjunto $\mathcal{C}$ se comporta durante o avanço de tempo. Inicialmente, $\mathcal{C}$ está vazio e não contém nenhum componente. Com o passar

do tempo, componentes começam a aparecer. Cada componente começa como uma única localidade p, que foi tocada por um conjunto S de usuários, com $|S| \geq M$. Com o aumento do instante de tempo, o conjunto de localidades alcançáveis pelos usuários em S localmente se parece como uma bola de crescimento uniforme com o centro em p (note que o conjunto $\bigcap_{j \in S} B(j,t)$ de localidades alcançáveis por todos os usuários em S pode parecer como uma união de várias bolas, assim como o mesmo conjunto S pode dar origem a múltiplos componentes de $\mathcal{C}$). Enquanto essas bolas crescem, duas ou mais eventualmente podem se tocar, dando origem a um único componente. Em geral, pares de componentes estarão se conectando e juntando. Portanto, em qualquer instante de tempo, cada componente C de $\mathcal{C}$ pode ser visto como uma união de bolas que crescem uniformemente, com diferentes raios, cada qual foi iniciada como um componente independente numa única localidade p. Note que, como os fantasmas crescem indefinidamente, em algum instante de tempo o conjunto $\mathcal{C}$ irá ser formado por apenas um único componente.

Antes de apresentar a função de cost-sharing, vamos mostrar uma correspondência entre o modo como o conjunto $\mathcal{C}$ cresce e o algoritmo primal-dual padrão para a árvore de Steiner (ver Algoritmo 2.2 do Capítulo 2), que será útil mais tarde. O algoritmo mantém uma lista de componentes. Inicialmente, cada vértice terminal está em um componente separado. O algoritmo então cresce uma bola uniformemente ao redor de cada vértice e no instante em que as duas bolas se tocam, é construído (pago) o caminho que liga esses dois vértices, juntando os dois componentes em um só. A única diferença é que, enquanto o algoritmo da árvore de Steiner começa com um conjunto fixo de componentes, no processo fantasma os componentes podem ser criados em pontos quaisquer (pontos onde uma localidade foi tocada por pelo menos M usuários) durante o tempo. Iremos fazer uso dessa correspondência para mostrar que os cost-shares pagam por uma fração constante do custo da árvore de Steiner na solução que iremos construir.

Seja $R(C)$ o raio de um componente $C \in \mathcal{C}$. O Algoritmo 6.2 descreve o crescimento fantasma para o Rent-or-Buy.

Algoritmo 6.2: Fantasma-RorB

Entrada: grafo $G = (V, E)$, demandas D;
Saída: $\tau(j), \tau'(j)$ para todo $j \in D$;
início
  $\mathcal{C} \leftarrow \emptyset$;
  $t \leftarrow 0$;
  $\tau(j) \leftarrow 0$ para todo $j \in D$; `/* tempo em que` j `se conecta */`
  $\tau'(j) \leftarrow 0$ para todo $j \in D$; `/* tempo em que` j `fica satisfeito */`
  enquanto $\exists j \in D$ *não conectado e* $\mathcal{C}$ *está disconexo* **faça**
    Seja $S_p = \{j \in D | t \geq c_{j,p}\}$ para alguma facilidade p;
    aumente t (e aumente $R(C), \forall C \in \mathcal{C}$ baseado no crescimento de t) até que:
      (a) $|S_p| \geq M$, ou
      (b) $R(C) \cap R(C') \neq \emptyset$
    se *(a)* **então**
      $j \in S_p$ fica conectado a p;
      se $\tau(j) = 0$ **então** $\tau(j) \leftarrow t$; `/*` j `se conecta */`
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$;
      $R(\{p\}) \leftarrow 0$;
    se *(b)* **então**
      Seja I as arestas do menor caminho entre C e C';
      junte C, C' e I , dando origem a um único componente C'';
      se *a raiz* $r \in C''$ **então**
        para todo $j \in D$ *conectado uma facilidade* $p \in C''$ **faça**
          se $\tau'(j) = 0$ **então** $\tau'(j) \leftarrow t$; `/*` j `fica satisfeito */`
  devolva $\tau(j), \tau'(j)$ *para todo* $j \in D$
fim

6.2.2 Função de cost-sharing

Antes de mostrar como são calculados os cost-shares, faremos algumas definições. Dizemos que um usuário j no instante de tempo t está *conectado* a um componente C de $\mathcal{C}$ se $B(j, t) \cap C \neq \emptyset$. O usuário está *satisfeito* se está conectado a um componente que contém o vértice raiz r. Para cada usuário j, $\tau(j)$ é o instante de tempo em que o usuário j se conecta a algum componente (o primeiro componente a que ele se conecta) e seja $\tau'(j)$ o instante de tempo em que j fica satisfeito.

Iremos fazer cada usuário pagar pelo custo de conexão associado a ele. Isto sugere que

cobremos de cada usuário uma quantidade proporcional ao tempo que o usuário passou desconectado. Além disso, cada usuário conectado a um componente C deve contribuir com uma parte do custo de crescimento do componente. Se um usuário está conectado a vários componentes, deixamos que este contribua somente com o componente o qual sua contribuição é menor.

Para cada componente C de $\mathcal{C}$, seja $D(C)$ o conjunto de usuários conectados a C. Para um usuário conectado j, seja $a_j(t)$ o tamanho máximo $|D(C)|$ sobre todos os componentes aos quais j está conectado no instante de tempo t. Vamos definir uma função $f_j(t)$ para cada usuário j como segue: $f_j(t) = 1$ para $0 \leq t \leq \tau(j)$, $f_j(t) = M/a_j(t)$ para $\tau(j) \leq t \leq \tau'(j)$, e $f_j(t) = 0$ para $t > \tau'(j)$. Definiremos duas versões dos custos compartilhados α_j e α'_j para cada usuário j. A função de cost-sharing final, determinada no Teorema 6.19, será uma soma ponderada dos custos compartilhados que vamos definir agora, balanceada para otimizar a fração do custo recuperado. Os custos compartilhados α e α' são definidos como:

$$\alpha_j = \int_0^{\tau'(j)} f_j(t)dt \quad \text{e} \quad \alpha'_j = \int_0^{\tau(j)} f_j(t)dt = t(j).$$

A propriedade cross-monotonic é fácil de ser obtida, os detalhes serão omitidos, sendo estes análogos à prova desta propriedade para o caso do Facility Location.

Teorema 6.10. *Os custos compartilhados α e α' são funções cross-monotonic.*

Esboço. Ao adicionar mais usuários, fazemos o conjunto $\mathcal{C}$ maior e portanto cada usuário se conecta antes. Além disso, cada componente de $\mathcal{C}$ só aumenta quando aumentamos o número de usuários, assim como aumenta o número de usuários conectados a um componente, portanto os cost-shares diminuem. ■

6.2.3 O processo real: construindo uma solução

Cada novo componente do conjunto $\mathcal{C}$ começa como sendo um único ponto p. Iremos chamar esses pontos de *centros*. Para uma localidade p, seja $t(p)$ o instante de tempo em que p aparece no conjunto $\mathcal{C}$. Note que $\mathcal{C}$ em um dado instante de tempo t é a união de bolas $B(p, t - t(p))$, uma bola de raio $t - t(p)$ ao redor de cada centro p.

Queremos usar esses centros como pontos de agrupamento. Contudo, para conseguirmos pagar pela árvore de Steiner nos centros, devemos selecionar somente um subgrupo de centros para abrir, e atribuir cada usuário de tal forma que este contribua com no máximo um centro. Faremos isso de uma maneira parecida com que fizemos no Facility Location.

Para cada centro p, decidimos se este será aberto no instante em que o centro é criado. Declaramos p aberto se, no instante de tempo $t(p)$ não há nenhum centro aberto dentro

de um raio de comprimento $2t(p)$ de p. Assim como no caso do Facility Location, isso garante que nenhum usuário pode estar no conjunto $S_p = \{j \in D | c_{j,p} \leq t(p)\}$ dos usuários que contribuem para criar p para dois centros abertos p distintos. O seguinte resultado segue diretamente do lema correspondente da seção do Facility Location.

Lema 6.11. *Seja p um centro aberto, e seja $j \in S_p$. Então $3\alpha'_j \geq t(p)$. Para um usuário j que não pertence a nenhum S_p, há um centro aberto p tal que $c_{j,p} \leq 3\alpha'_j$.*

Assim, iremos construir a rede da seguinte forma: compramos a árvore de Steiner que conecta centros abertos a r (raiz) e conectamos cada usuário a seu centro aberto mais próximo. O Lema 6.11 diz que os custos compartilhados α' podem pagar por 1/3 do custo de conexão da rede. A próxima seção tem como objetivo mostrar que os cost-shares α podem pagar por uma fração constante do custo da árvore que construímos.

6.2.4 Análise

Limitando o custo da árvore de conexão

Vamos assumir, para propósitos de análise do algoritmo, que o Algoritmo 2.2 é usado para construir a árvore de Steiner nos centros abertos. O algoritmo começa com cada centro em um componente separado. Uma bola cresce uniformemente durante o tempo em torno de cada centro. Toda vez que duas bolas se tocam, verificamos se os centros dessas bolas estão no mesmo componente. Se não estão, um caminho mínimo entre os centros das bolas é comprado, juntando os dois componentes em um só. Além disso, o centro p da primeira bola a alcançar o vértice raiz r pode comprar um caminho mínimo entre p e r.

Podemos pensar que o algoritmo paga custo M por unidade de tempo para crescer cada componente que não contém o vértice raiz r. (O componente que contém r cresce sem custo, i.e., crescemos este componente de graça). Isto é, se $a(t)$ é o número de componentes não-raiz no instante de tempo t, o custo total de crescer os componentes é $M \int_0^\infty a(t)dt$, onde a integral é sobre todo o tempo de execução do algoritmo. Pelo Teorema 2.8 do Capítulo 2, o custo da árvore construída (solução primal) é no máximo duas vezes o custo de crescimento (solução dual).

Para um centro aberto p, seja $S_p(t)$ o conjunto de usuários que estão a uma distância de no máximo t de p. Note que $S_p = S_p(t(p))$. Seja C' um componente de Steiner num dado instante de tempo t. Definimos um conjunto (que varia com o tempo) Contrib(C') de usuários, tal que

$$\text{Contrib}(C') = \bigcup_{p \in C'} S_p(\max\{t, t(p)\}).$$

O lema a seguir diz que nenhum usuário contribui para mais de um componente em qualquer instante de tempo.

Lema 6.12. *Em qualquer instante de tempo t, os conjuntos contribuintes de dois componentes de Steiner distintos C'_1, C'_2 são disjuntos.*

Demonstração. Seja j um usuário que, no instante de tempo t, está contribuindo para ambos centros abertos p e q ao mesmo tempo. Então $c_{j,p} \leq t$ e $c_{j,q} \leq t$ e portanto, pela desigualdade triangular, $c_{p,q} \leq 2t$. Mas p e q têm ambos um raio de tamanho t que começou crescendo desde o início do algoritmo, ou seja, os raios já se tocaram, pois $c_{p,q} \leq 2t$. Portanto, como p e q são centros abertos, deve-se comprar o menor caminho entre eles e concluimos que p e q pertecem ao mesmo componente. ■

Considere um usuário j que pertence ao conjunto contribuinte de um componente de Steiner C'. No processo fantasma, este usuário pode estar conectado a vários componentes de $\mathcal{C}$, e alguns desses componentes podem ter um grande número de usuários conectados a eles. Então mesmo se o componente C' é pequeno, i.e., não tem muitos contribuintes, pode acontecer de que a maioria dos seus contribuintes tenham todos uma conexão com um componente grande de $\mathcal{C}$, e portanto suas contribuições $f_j(t)$ não são suficientes para suportar o crescimento do componente C' no instante de tempo t. No entanto, é possível mostrar que no tempo $3t$ o componente C' vai ter crescido suficiente para cobrir a área que os grandes componentes originais ocupavam no instante de tempo t. Juntos, esses usuários podem pagar o suficiente para o crescimento de C' no instante de tempo $3t$ com suas taxas de crescimento $f_j(t)$ no instante de tempo t, que é o que diz o seguinte lema.

Lema 6.13. *Seja j um usuário em um instante de tempo t, $j \in Contrib(C')$ para algum componente de Steiner C'. Então $f_j(t/3) \geq M/|Contrib(C')|$.*

Antes de provar o Lema 6.13, vamos mostrar alguns resultados úteis que ajudarão na prova do lema.

Proposição 6.14. *Para uma localidade $p \in C$, há um centro aberto q (possivelmente $q = p$) tal que $c_{p,q} \leq 2t(p)$.*

Demonstração. Se p está aberto, $q = p$ e claramente a desigualdade vale. Caso contrário, pela nossa decisão de quais centros abrir, existe um centro q que foi aberto antes, tal que $c_{p,q} \leq 2t(p)$. ■

Para o lema seguinte, vale lembrar que um usuário i que está conectado a um componente C não está necessariamente contribuindo com C, pois a bola de C pode estar fazendo intersecção com a bola de i, mas i ainda não alcançou algum centro aberto $p \in C$.

Lema 6.15. *Suponha que num dado instante de tempo t, usuários i e j estão conectados no mesmo componente C do conjunto $\mathcal{C}$. Então, no instante de tempo $3t$, i e j pertencem ao conjunto contribuinte do mesmo componente de Steiner C'.*

Demonstração. Sejam i e j usuários conectados a um componente C de $\mathcal{C}$ em um dado instante de tempo t. Note que C é uma união de bolas ao redor de centros (abertos ou não abertos). Se o usuário i está conectado com C, então existe um centro q tal que a bola $B(q, t - t(q))$ de q tem intersecção não vazia com o fantasma de i. Da mesma forma, existe um centro q' tal que $B(q', t - t(q'))$ de q' tem intersecção não vazia com o fantasma de j. Como os centros q e q' pertencem ao mesmo componente, existe uma sequência de centros $q_1 = q, q_2, \ldots, q_k = q'$, com bolas $B_l(q_l, t - t(q_l))$ tal que as bolas de todos pares de centros consecutivos desta sequência se interceptam.

Pela Proposição 6.14 existe uma sequência de centros abertos (não necessariamente distintos) $p_1, \ldots, p_k$ tal que para cada l, $c_{p_l,q_l} \leq 2t(q_l)$. Como exemplo, veja a Figura 6.5. É fácil de verificar que a distância $c_{i,p_1} \leq 2t(q_1) + t \leq 3t$ e analogamente $c_{j,p_k} \leq 3t$. A distância $c_{p_l,p_{l+1}}$ pode ser limitada por $4t$ da seguinte forma:

$$\begin{aligned} c_{p_l,p_{l+1}} &\leq c_{p_l,q_l} + c_{q_l,q_{l+1}} + c_{q_{l+1},p_{l+1}} \\ &\leq 2t(q_l) + (t - t(q_l)) + (t - t(q_{l+1})) + 2t(q_{l+1}) \\ &\leq 4t. \end{aligned}$$

Portanto, cada par de bolas consecutivas $B(p_l, 3t)$ devem se interceptar, provando que p_1 e p_k estão no mesmo componente de Steiner no instante de tempo $3t$. Pela definição de Contrib, ambos i e j devem estar no conjunto contribuinte desse componente de Steiner. ■

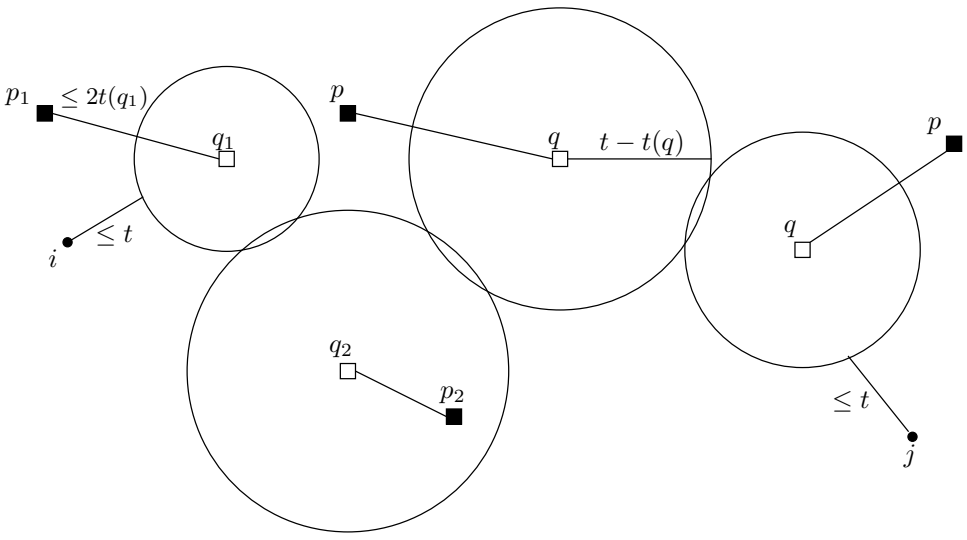

Figura 6.5: Clientes i e j conectados a um componente C no instante de tempo t

Agora estamos prontos para provar o Lema 6.13.

Demonstração do Lema 6.13. Como $|S_p| \geq M$ para qualquer centro aberto p, $|\text{Contrib}(C')|$ é sempre pelo menos M, e portanto $M/|\text{Contrib}(C')| \leq 1$. Se j não está conectado no instante de tempo $t/3$, então $f_j(t/3) = 1$ e a desigualdade segue trivialmente, pois $1 = f_j(t/3) \geq M/|\text{Contrib}(C')|$. Se j está conectado no instante de tempo $t/3$, $f_j(t/3) = M/|D(C)|$, para algum componente C de $\mathcal{C}$, lembrando que $D(C)$ é o conjunto dos usuários conectados a C. Pelo Lema 6.15, no instante de tempo t $\text{Contrib}(C')$ contém todos usuários que estavam contribuindo para C no instante de tempo $t/3$. ■

Usando o Lema 6.13, limitamos o custo da árvore construída.

Lema 6.16. *O custo da árvore de conexão contruída é no máximo* $6\sum_{j\in D}\alpha_j$.

Demonstração. Em qualquer instante de tempo t, os usuários obtêm um rendimento de $Ma(t)$ de suas contribuições no tempo $t/3$. Afirmamos que $Ma(t) \leq \sum_{j\in D} f_j(t/3)$. Para ver isso, seja k o número de componentes de Steiner num dado instante de tempo t. Os componentes são enumerados $C_1, C_2, \ldots, C_k$. Pelo Lema 6.13 , $f_j(t/3) \geq M/|\text{Contrib}(C')|$. Assim temos

$$\begin{aligned}
\sum_{j\in D} f_j(t/3) &= \sum_{j\in C_1} f_j(t/3) + \ldots + \sum_{j\in C_k} f_j(t/3) \\
&\geq \sum_{j\in C_1} \frac{M}{|\text{Contrib}(C_1)|} + \ldots + \sum_{j\in C_k} \frac{M}{|\text{Contrib}(C_k)|} \\
&= \frac{M|\text{Contrib}(C_1)|}{|\text{Contrib}(C_1)|} + \ldots + \frac{M|\text{Contrib}(C_k)|}{|\text{Contrib}(C_k)|} \\
&= M + \ldots + M \\
&= Mk = Ma(t).
\end{aligned}$$

Portanto

$$\begin{aligned}
\int_0^\infty Ma(t) &\leq \int_0^\infty \sum_{j\in D} f_j(t/3)dt \\
&= \sum_{j\in D} \int_0^\infty f_j(t/3)dt \\
&= \sum_{j\in D} 3\alpha_j \\
&= 3\sum_{j\in D} \alpha_j.
\end{aligned}$$

Como afirmamos anteriormente, a quantidade $M\int_0^\infty a(t)dt$ pode pagar por metade do custo da árvore e portanto o custo de Steiner da nossa solução é no máximo $6\sum_{j\in D}\alpha_j$. ■

Recuperação do custo e competitividade

Teorema 6.17 (Recuperação de custo)**.** *O custo da solução construída é no máximo* $\sum_{j \in D}(3\alpha'_j + 6\alpha_j)$.

Demonstração. Pelo Lema 6.11, os custos de conexão podem ser limitados por $3\sum_{j \in D} \alpha'_j$. Pelo Lema 6.16, os custos da árvore de conexão podem ser limitados por $6\sum_{j \in D} \alpha_j$. Como o custo da solução é igual ao custo de conexão mais o custo de Steiner, então o lema segue. ■

Teorema 6.18 (Competitividade)**.** *Toda solução viável para uma instância do jogo rent-or-buy (em particular, uma solução ótima) tem custo pelo menos*

$$\max\left(1/2\sum_{j \in D}\alpha_j, \sum_{j \in D}\alpha'_j\right).$$

Demonstração. Considere uma solução qualquer O de uma instância do Rent-or-Buy. Iremos mostrar que seu custo deve ser maior ou igual a $\frac{1}{2}\sum_{j \in D}\alpha_j$ (a prova para $\sum_{j \in D}\alpha'_j$ é mais simples e é omitida).

Suponha que temos distâncias $d_{j,e}$ de arestas para cada usuário j. Em outras palavras, cada usuário j "enxerga" um custo para a aresta e, i.e., sob perspectiva do usuário j, a aresta e tem custo $d_{j,e}$. Além disso, supomos que as distâncias $d_{j,e}$ tenham as seguintes propriedades:

1. Para cada aresta e e cada usuário j, $d_{j,e} \leq c_e$.
2. Para cada aresta e, $\sum_{j \in J} d_{j,e} \leq 2Mc_e$.
3. O menor caminho de todo usuário j à raiz r na métrica $d_{j,e}$ (para todas arestas e pertecentes ao caminho) tem comprimento pelo menos α_j.

Novamente, considere uma solução arbitrária O. Como cada usuário tem um caminho para a raiz r, pela propriedade 3, temos

$$\alpha_j \leq \sum_{e \text{ comprada}} d_{j,e} + \sum_{e \text{ alugada para } j} d_{j,e},$$

lembrando que as arestas compradas são as arestas que pertencem à árvore de Steiner (pagando M vezes o seu custo) e as arestas alugadas são as arestas de conexão.

Somando a desigualdade acima sobre todos $j \in D$, obtemos

$$\sum_{j \in D}\alpha_j \leq \sum_{e \text{ comprada}}\sum_{j \in D} d_{j,e} + \sum_{j \in D}\sum_{e \text{ alugada para } j} d_{j,e},$$

O custo da solução arbitrária O pode ser escrito como

$$\text{val}(O) = \sum_{e \text{ comprada}} Mc_e + \sum_{j \in D} \sum_{e \text{ alugada para } j} c_e.$$

Pelas propriedades 1 e 2, temos que

$$\sum_{e \text{ comprada}} \sum_{j \in D} d_{j,e} \leq 2 \sum_{e \text{ comprada}} Mc_e$$
$$\sum_{j \in D} \sum_{e \text{ alugada para } j} d_{j,e} \leq \sum_{j \in D} \sum_{e \text{ alugada para } j} c_e.$$

Combinando essas desigualdades, obtemos $\sum_{j \in D} \alpha_j \leq 2 \cdot \text{val}(O)$.

Nos resta agora mostrar que a métrica $d_{j,e}$ existe de fato. Para um usuário j, seja $S_j(t)$ o conjunto de localidade *alcançáveis* por j no instante de tempo t, i.e., uma localidade p é alcancável por j:

(i) se p está dentro da bola fantasma de j (ou seja, $c_{j,p} \leq t$), ou

(ii) p está em algum componente ao qual j está conectado.

Uma aresta e é *cruzada* pelo conjunto $S_j(t)$ se e tem uma ponta dentro de S_j e a outra fora. Começamos com todos $d_{j,e} = 0$. Em cada instante de tempo t, aumentamos a distância $d_{j,e}$ de cada aresta e que é cruzada por $S_j(t)$ à taxa $f_j(t)$.

Em cada instante de tempo aumentamos a distância de todas arestas em algum $j - s$ corte (i.e., um corte que separa j de s) pois qualquer $S_j(t)$ é um $j - s$ corte. A distância das arestas do $j - s$ corte é aumentada a taxa de $f_j(t)$, assim, a distância de um caminho mínimo entre j e s é igual a $\int_0^\infty f_j(t)dt = \alpha_j$, em outras palavras, claramente a menor distância entre j e s é o raio α_j que cresceu em j desde o início até o fim do algoritmo. Assim, a propriedade 3 está provada. A fronteira de cada S_j se move a velocidade 1 ou mais rápido no caso em que dois componentes se juntam. Assim, nenhuma aresta vai acumular $d_{j,e}$ que ultrapasse o seu comprimento inicial c_e, pois $d_{j,e}$ vai sendo acrescido na mesma velocidade do crescimento do componente e quando os "saltos" de junção de componentes acontecem, ou o salto é pequeno suficiente para crescer menos que o comprimento total c_e ou o salto ultrapassa c_e, não sendo contabilizado nos $d_{j,e}$. Desta forma, a propriedade 1 está provada. Por último, vamos considerar a propriedade 2. Uma localidade na aresta e não pode ser cruzada por duas fronteiras de componentes, pois senão os componentes já teriam se juntado e formado um só. Portanto, uma localidade em e só pode ser cruzada por uma fronteira de componente. Da mesma forma, uma localidade na aresta só pode ser cruzada por uma fronteira com no máximo M fantasmas de usuários, pois se houvesse mais de M fronteiras de fantasmas, já teria se transformado

em fronteira de componente (lembrando que mais que M usuários agrupados formam um componente). Portanto, em uma localidade em e podemos ter no máximo uma fronteira de M fantasmas de usuários e uma fronteira de componente (pagando M vezes o custo da aresta). Assim $\sum_{j \in D} d_{j,e} \leq Mc_e + Mc_e = 2Mc_e$, provando a propriedade 2. ■

Teorema 6.19. *A função de cost-share $\xi(D, j) = 1/5\alpha'_j + 2/5\alpha_j$ é cross-monotonic, competitiva e recupera $1/15$ do custo da solução construída.*

Demonstração. Sejam $\text{val}(O, D)$ e $\text{val}(O^*, D)$ o custo da solução O construída e o custo de uma solução ótima O^*, respectivamente, sobre o conjunto de usuários D. O teorema afirma que

$$\frac{1}{15}\text{val}(O, D) \leq \sum_{j \in D} \left(\frac{1}{5}\alpha'_j + \frac{2}{5}\alpha_j \right) \leq \text{val}(O^*, D).$$

A primeira desigualdade acima é válida pois, multiplicando essa desigualdade por 15, temos

$$\text{val}(O, D) \leq \sum_{j \in D} \left(\frac{15}{5}\alpha'_j + \frac{30}{5}\alpha_j \right) = \sum_{j \in D} \left(3\alpha'_j + 6\alpha_j \right),$$

que é válida segundo o Teorema 6.17.

Para a segunda desigualdade, temos que provar que

$$\text{val}(O^*, D) \geq \sum_{j \in D} \left(\frac{1}{5}\alpha'_j + \frac{2}{5}\alpha_j \right).$$

Mas sabemos, pelo Teorema 6.18 que $\text{val}(O^*, D) \geq \max(1/2 \sum_{j \in D} \alpha_j, \sum_{j \in D} \alpha'_j)$. Então, é suficiente provar que

$$\max \left(1/2 \sum_{j \in D} \alpha_j, \sum_{j \in D} \alpha'_j \right) \geq \sum_{j \in D} \left(\frac{1}{5}\alpha'_j + \frac{2}{5}\alpha_j \right).$$

Temos dois casos a analisar, dependendo do valor retornado pela função de máximo.

(i) Se $1/2 \sum_{j \in D} \alpha_j \geq \sum_{j \in D} \alpha'_j$ então

$$\begin{aligned}
\text{val}(O^*, D) &\geq \frac{1}{2} \sum_{j \in D} \alpha_j \\
&= \frac{4}{10} \sum_{j \in D} \alpha_j + \frac{1}{10} \sum_{j \in D} \alpha_j \\
&\geq \frac{4}{10} \sum_{j \in D} \alpha_j + \frac{2}{10} \sum_{j \in D} \alpha'_j \\
&= \frac{2}{5} \sum_{j \in D} \alpha_j + \frac{1}{5} \sum_{j \in D} \alpha'_j,
\end{aligned}$$

onde a última desigualdade vem do fato que neste caso estamos considerando $1/2 \sum_{j \in D} \alpha_j \geq \sum_{j \in D} \alpha'_j$.

(ii) Caso contrário, se $1/2 \sum_{j \in D} \alpha_j < \sum_{j \in D} \alpha'_j$ então

$$\begin{aligned} \text{val}(O^*, D) &\geq \sum_{j \in D} \alpha'_j \\ &= \frac{4}{5} \sum_{j \in D} \alpha'_j + \frac{1}{5} \sum_{j \in D} \alpha'_j \\ &\geq \frac{2}{5} \sum_{j \in D} \alpha_j + \frac{1}{5} \sum_{j \in D} \alpha'_j, \end{aligned}$$

onde a última desigualdade vem do fato que neste caso estamos considerando $1/2 \sum_{j \in D} \alpha_j < \sum_{j \in D} \alpha'_j$.

■

6.3 Connected Facility Location

6.3.1 O processo fantasma

Para um dado conjunto D de usuários, iremos executar o processo fantasma para determinar três diferentes custos compartilhados $\alpha_j, \alpha'_j, \alpha''_j$ para todo $j \in D$. A função de cost-sharing final de j será uma combinação desses três.

Assim como nos outros problemas, associamos uma noção de tempo com o processo. Para cada demanda j, temos a bola fantasma $B(j,t)$ que tem centro em j e raio com tamanho do tempo atual t. Quando M ou mais bolas se interceptam numa localidade p em comum, nós abrimos p e a partir de então p é chamado de *centro* (lembrando que no processo fantasma abrimos todos os centros, no processo real, escolhemos um subgrupo de centros para abrir). Usamos $t(p)$ para referenciar o tempo em que o centro p é aberto e D_p o conjunto de usuários responsável por abrir p, i.e., todas demandas j que satisfazem $c_{j,p} \leq t(p)$. Dizemos que as demandas em D_p formam um *aglomerado*.

Aqui, temos novamente a noção de componentes, idêntica à Seção 6.2.1, onde as primeiras localidades pertencentes a $\mathcal{C}$ são pontos onde M ou mais bolas estão se interceptando e os componentes de $\mathcal{C}$ aparecem em instantes diferentes de tempo, exatamente como já haviamos descrito na Seção 6.2.1.

Aqui, novamente $\tau(j), \tau'(j)$ denotam respectivamente o instante de tempo em que um usuário j se conectou pela primeira vez a algum componente e o instante de tempo em que j ficou satisfeito. Lembrando que j fica satisfeito quando se conecta a algum componente que contém o vértice raiz r.

No instante de tempo t, a contribuição do usuário j ao custo de abertura de uma facilidade i é $\max(0, t - c_{j,i})$. Se a contribuição total para a facilidade i é igual ao custo de abertura f_i, então i é declarada aberta. Seja t_i o tempo em que a facilidade i se tornou aberta e D_i o conjunto de demandas que contribuem para abrir i no instante de tempo t_i.

6.3.2 A função de cost-sharing

Para cada usuário $j \in D$, definimos três custos compartilhados diferentes:

$$\alpha_j = \tau(j),$$
$$\alpha'_j = \tau(j) + M \cdot \int_{\tau(j)}^{\tau'(j)} \frac{1}{a_j(t)} dt, \text{e}$$
$$\alpha''_j = \min\Big(\min_{i:j \in D_i} t_i, \min_{i:j \notin D_i} c_{j,i} \Big).$$

Relembrando que $a_j(t)$ é o tamanho máximo $|D(C)|$ sobre todos os componentes aos quais j está conectado no instante de tempo t, onde $D(C)$ é o conjunto de usuário conectados ao componente C.

Note que os primeiros dois custos compartilhados são idênticos aos custos compartilhados utilizados no problema Rent-or-Buy, e o terceiro custos compartilhados é idêntico ao custos compartilhados utilizado no problema Facility Location. Assim como fizemos no Rent-or-Buy, vamos fazer uma soma ponderada desses custos compartilhados para obter a função de cost-sharing final.

Teorema 6.20. *Os custos compartilhados α, α' e α'' são funções cross-monotonic.*

Demonstração. (Veja também Teoremas 6.2 e 6.10) Considere um conjunto D' de usuários e vamos assumir que adicionamos um novo usuário k a D'. Ao adicionar o usuário k, em qualquer instante de tempo t, o conjunto $\mathcal{C}$ só pode se tornar maior. Portanto o tempo de conexão $\tau(j)$ de uma demanda $j \in D'$ pode somente ficar menor. Deste modo, como o número de usuários que estão conectados a um componente C de $\mathcal{C}$ no instante de tempo t só pode aumentar, então $a_j(t)$ também só pode aumentar, para um usuário $j \in D'$. Além disso, ao adicionar k a D', o tempo de abertura t_i de qualquer facilidade i só pode diminuir. ■

6.3.3 O processo real: algoritmo

Nós executamos o processo fantasma, mas também levamos em consideração as seguintes regras para decidir quais localidades e facilidades serão realmente abertas.

- Abrimos a localidade p no instante de tempo t_p se não há outra localidade aberta q tal que $c_{p,q} \leq 2t_p$.

- Abrimos a facilidade i no instante de tempo t_i se não há outra facilidade aberta k tal que $c_{i,k} \leq 2t_i$.

As localidades que são abertas pelas regras acima são chamadas de *centros*. Seja F' o conjunto de facilidades que foram abertas. Como será visto abaixo, essas regras garantem que

(i) todos aglomerados D_p com centro p são disjuntos, e

(ii) todos conjuntos D_i com $i \in F'$ são disjuntos.

A solução final é construída como segue. Para cada aglomerado D_p em que p está aberto, determinamos uma facilidade $i(p)$ que é a facilidade aberta mais próxima de p. Dizemos que $i(p)$ é a facilidade do aglomerado D_p. Seja F o conjunto de todas as facilidades $i(p)$ que correspondem a aglomerados D_p abertos (i.e., aglomerados cujos centros estão abertos), ou seja, $F = \{i \in F' | i = i(p) \text{ para algum centro } p\}$. Atribuímos todas as demandas de um aglomerado D_p à sua facilidade $i(p)$. Usuários que não estão contidos em nenhum aglomerado aberto são atribuídos à facilidade de seu centro mais próximo.

Construímos uma árvore de Steiner nos centros e para cada centro p compramos o caminho mínimo (i.e., pagamos custo M vezes o custo do caminho) de p à facilidade $i(p)$ do aglomerado. Isto garante que as facilidades em F são conectadas. Observe que nenhum usuário é atribuído a alguma facilidade em $F' \setminus F$, e então conseqüentemente fechamos todas as facilidades em $F' \setminus F$.

6.3.4 Análise

Lema 6.21. *Para quaisquer dois centros p e q, D_p e D_q são disjuntos.*

Demonstração. Esse lema tem o mesmo enunciado do Lema 6.12, mas adaptado para o problema Connected Facility Location. A prova deste é idêntica a prova do Lema 6.12. ■

Lema 6.22. *Para quaisquer duas facilidades abertas i e k, D_i e D_k são disjuntos.*

Demonstração. Assumindo que abrimos todas as facilidades em F', podemos provar o lema de maneira análoga à prova do Lema 6.12. Como $F \subseteq F'$, o lema segue também para as facilidade em F. ■

Lema 6.23. *(i) Suponha que $j \in D_p$, para algum centro p. Então $t_p \leq 3\alpha_j$. (ii) Para cada usuário j que não pertence a algum agrupamento aberto, há um centro p tal que $c_{j,p} \leq 3\alpha_j$.*

Demonstração. A parte (i) do lema vem direto do Lema 6.5 e a parte (ii) vem direto do Lema 6.6. ■

Lema 6.24. *Suponha que $j \in D_i$ para alguma facilidade $i \in F'$. Então $t_i \leq 3\alpha''_j$. Para cada usuário j que não pertence a nenhum conjunto D_k com facilidade $k \in F'$, há uma facilidade $i \in F'$ tal que $c_{i,j} \leq 3\alpha''_j$.*

Demonstração. A prova é a mesma do Lema 6.23. ■

Lema 6.25. *O custo da árvore de Steiner em todos os centros é no máximo $6\sum_{j\in D}\alpha'_j$.*

Demonstração. Análoga à prova do Lema 6.16, tomando o cuidado de que as notações α e α' foram invertidas nos dois problemas. ■

Lema 6.26. *O custo de abrir uma facilidade $i \in F'$ é no máximo $\sum_{j\in Q_i}(3\alpha''_j - c_{i,j})$.*

Demonstração. Temos que $f_i = \sum_{j\in D_i}(t_i - c_{i,j})$. A prova segue usando o Lema 6.24. ■

Lema 6.27. *O custo de conectar todas usuários de um agrupamento aberto D_p à facilidade $i(p)$ é no máximo $\sum_{j\in D_p}\left(6\alpha_j + 3\alpha''_j\right)$.*

Demonstração. Seja $j \in D_p$. Se existe alguma facilidade aberta $i \in F'$ tal que $j \in D_i$, temos que $c_{j,i} \leq t_i \leq 3\alpha''_j$, pelo Lema 6.24. Caso contrário, se j não está contido em nenhum D_k com facilidade $k \in F'$, pelo Lema 6.24 existe uma facilidade aberta $i \in F'$ tal que $c_{j,i} \leq 3\alpha''_j$. Ou seja, para cada $j \in D_p$, há uma facilidade em F' com distância no máximo $3\alpha''_j$. Como nós escolhemos $i(p)$ de F' como a facilidade que está mais próxima de p, temos

$$c_{p,i(p)} \leq t_p + \min_{l\in D_p}(3\alpha''_l).$$

Portanto,

$$c_{j,i(p)} \leq c_{j,p} + c_{p,i(p)} \leq t_p + t_p + \min_{l\in D_p}(3\alpha''_l) \leq 2t_p + 3\alpha''_j \leq 6\alpha_j + 3\alpha''_j,$$

onde a última desigualdade vem do Lema 6.23. ■

Seja $L \subseteq D$ o conjunto de usuários que não está contido em nenhum agrupamento aberto.

Lema 6.28. *O custo de conectar todas os usuários que não estão conectadas a nenhum agrupamento aberto à facilidade do seu centro mais próximo é no máximo $\sum_{j\in L}(6\alpha_j + 3\alpha''_j)$.*

Demonstração. Seja $j \in L$. Por argumentos parecidos aos da prova do Lema 6.27, podemos encontrar uma facilidade $i \in F'$ tal que $c_{j,i} \leq 3\alpha''_j$. Além disso, pelo Lema 6.23 existe um centro aberto p tal que $c_{j,p} \leq 3\alpha_j$. Seja $L(p)$ os usuários em L cujo centro mais próximo é p. Para um usuário $l \in L(p)$, temos que $c_{p,i} \leq c_{l,p} + c_{l,i} \leq 3\alpha_l + 3\alpha''_l$, mas como escolhemos $i(p)$ sendo a facilidade mais próxima a p, então para $c_{p,i(p)} \leq c_{p,i}$, e isso vale para qualquer $l \in L(p)$. Portanto

$$c_{p,i(p)} \leq \min_{l \in L(p)} (3\alpha_l + 3\alpha''_l).$$

Então, concluímos que

$$\begin{aligned} c_{j,i(p)} &\leq c_{j,p} + c_{p,i(p)} \\ &\leq 3\alpha_j + \min_{l \in L(p)} (3\alpha_l + 3\alpha''_l) \\ &\leq 6\alpha_j + 3\alpha''_j. \end{aligned}$$

■

Lema 6.29. *O custo total de comprar o menor caminho entre um centro p e sua facilidade correspondente $i(p)$ é no máximo $\sum_{j \in D_p} (3\alpha_j + 3\alpha''_j)$.*

Demonstração. Cada aglomerado contém pelo menos M usuários. O custo de comprar o menor caminho entre p e $i(p)$ é

$$M \cdot c_{p,i(p)} \leq M \cdot (t_p + \min_{l \in D_p}(3\alpha''_l)) \leq \sum_{j \in D_p} \left(t_p + 3\alpha''_j\right) \leq \sum_{j \in D_p} \left(3\alpha_j + 3\alpha''_j\right),$$

onde a primeira desigualdade vem da prova do Lema 6.27 e a última desigualdade segue do Lema 6.23. ■

Teorema 6.30 (Recuperação de Custo)**.** *O custo da solução construída é no máximo $\sum_{j \in D} 9\alpha_j + 6\alpha'_j + 9\alpha''_j$.*

Demonstração. Direto dos lemas anteriores, somando custo de conexão, custo da árvore de Steiner, custo de abertura de facilidades e o menor caminho comprado entre um centro p e $i(p)$. ■

Teorema 6.31 (Competitividade)**.** *Toda solução viável para o problema CFL (em particular, uma solução ótima) em D tem custo pelo menos*

$$\max \left(\sum_{j \in D} \alpha_j, \frac{1}{2} \sum_{j \in D} \alpha'_j, \sum_{j \in D} \alpha''_j \right).$$

Demonstração. A prova dos limitantes inferiores para $\sum_{j\in D}\alpha_j$ e $\sum_{j\in D}\alpha'_j$ está no Teorema 6.18. Além disso, pelo Teorema 6.3, $\sum_{j\in D}\alpha''_j$ é um limitante inferior para o problema Facility Location mesmo quando as facilidade não devem estar conectadas. ■

Teorema 6.32. *A função de cost-sharing ξ, definida como*

$$\xi(D,j) = \frac{3}{10}\alpha_j + \frac{1}{5}\alpha'_j + \frac{3}{10}\alpha''_j$$

é cross-monotonic, competitiva e recupera $1/30$ *do custo da solução construída.*

Demonstração. Sejam $\text{val}(O,D)$ e $\text{val}(O^*,D)$ o custo da solução O construída e o custo de uma solução ótima O^*, respectivamente. O teorema afirma que

$$\frac{1}{30}\text{val}(O,D) \leq \sum_{j\in D}\left(\frac{3}{10}\alpha_j + \frac{1}{5}\alpha'_j + \frac{3}{10}\alpha''_j\right) \leq \text{val}(O^*,D).$$

A primeira desigualdade é válida pois, multiplicando essa desigualdade por 30, temos

$$\text{val}(O,D) \leq \sum_{j\in D}\left(9\alpha_j + 6\alpha'_j + 9\alpha_j\right),$$

que é válida segundo o Teorema 6.30.

Para a segunda desigualdade, temos que provar que

$$\text{val}(O^*,D) \geq \sum_{j\in D}\left(\frac{3}{10}\alpha_j + \frac{1}{5}\alpha'_j + \frac{3}{10}\alpha''_j\right).$$

Mas sabemos, pelo Teorema 6.31 que $\text{val}(O^*,D) \geq \max\left(\sum_{j\in D}\alpha_j, \frac{1}{2}\sum_{j\in D}\alpha'_j, \sum_{j\in D}\alpha''_j\right)$. Então, é suficiente provar que

$$\max\left(\sum_{j\in D}\alpha_j, \frac{1}{2}\sum_{j\in D}\alpha'_j, \sum_{j\in D}\alpha''_j\right) \geq \sum_{j\in D}\left(\frac{3}{10}\alpha_j + \frac{1}{5}\alpha'_j + \frac{3}{10}\alpha''_j\right).$$

Temos três casos a analisar, dependendo do valor retornado pela função de máximo. Os detalhes serão omitidos, mas são facilmente obtidos fazendo manipulações algébricas assim como feitas na demonstração do Teorema 6.19. ■

Capítulo 7

Técnica probabilística

Neste capítulo, vamos apresentar a idéia de compartilhamento do custo *esperado* de um algoritmo probabilístico, usando para isto o problema Rent-or-Buy. Para esta análise, usaremos novamente o Algoritmo 4.1, descrito no Capítulo 4. Usando este algoritmo, iremos definir funções probabilísticas de cost-sharing e provaremos que os custos esperados são cross-monotonic, competitivos, e com *alta probabilidade* recuperam pelo menos $\frac{1}{4}(1+\epsilon)^{-1}$ do custo da solução construída, onde $\epsilon > 0$ é uma constante arbitrária. Este capítulo está baseado nos resultados obtidos por Leonardi e Schaefer em [22].

O objetivo principal deste capítulo é demonstrar o uso de algoritmos probabilísticos para obtenção de funções de cost-sharing. Para tentar obter uma desaleatorização do Algoritmo 4.1, temos que calcular a função de cost-sharing esperada em tempo polinomial. Iremos fazer essa desaleatorização no Capítulo 8, que requer uma análise mais complexa do algoritmo, fugindo um pouco do objetivo deste capítulo, mas apresentando importantes resultados teóricos.

7.1 Rent-or-Buy

Para este capítulo, iremos assumir como verdadeiras as Suposições 3.1 (demandas unitárias) e 3.2 (raiz na solução).

7.1.1 Função de cost-sharing

Vamos aproximar a árvore de Steiner do Passo 2 do Algoritmo 4.1 usando uma árvore geradora mínima em F, que é computada utilizando o algoritmo primal-dual de Edmonds, apresentado no Algoritmo 5.2 no Capítulo 5, Seção 5.3. Lembrando, que pelos Teoremas 5.12 e 5.13, a função de cost-sharing para este problema é budget-balance e

cross-monotonic, e além disso, sabemos que um algoritmo ótimo para a árvore geradora mínima leva a uma 2-aproximação para o problema da árvore de Steiner.

Iremos definir então a função *auxiliar* de cost-sharing $\xi'(Q, j)$ probabilística para cada usuário $j \in Q$ como segue.

$$\xi'(Q,j) = \begin{cases} M \cdot \xi_{\text{MST}}(F,j) & \text{se } j \in F, \text{ e} \\ l(j,F) & \text{se } j \notin F, \end{cases}$$

onde $\xi_{\text{MST}}(F, j)$ é a função de cost-sharing do algoritmo de Edmonds (Algoritmo 5.2) para a árvore geradora mínima e $l(j, F)$ é o caminho mais curto de j à facilidade mais próxima em F. Note que tanto $\xi_{\text{MST}}(F, j)$ como $l(j, F)$ são variáveis aleatórias, que dependem do conjunto de facilidades $F \subseteq Q$ escolhido aleatoriamente no Passo 1 do Algoritmo 4.1.

A função de cost-sharing *final* ξ é simplesmente a função de cost-sharing auxiliar ξ' multiplicada por $\frac{1}{4}$, e é definida como:

$$\xi(Q,j) = \frac{1}{4}\text{E}[\xi'(Q,j)].$$

Note que poderíamos definir diretamente a função de cost-sharing ξ sem a necessidade de definí-la usando a função auxiliar ξ'. Não o fizemos por questões de clareza da notação nas demonstrações, como veremos a seguir.

7.1.2 Propriedades da função de cost-sharing

Iremos mostrar agora que a função de cost-sharing ξ é cross-monotonic. Seja D o conjunto de todas demandas (usuários). A idéia da prova é a seguinte. Sejam $Q' \subseteq D$ um subconjunto de demandas e $F' \subseteq Q'$ o conjunto de facilidades abertas pelo algoritmo em Q'. Vamos incluir em Q' uma demanda $k \notin Q'$. Chamaremos esse novo conjunto de demandas de $Q = Q' \cup \{k\}$. Usamos $F \subseteq Q$ para denotar o conjunto das facilidades abertas pelo algoritmo em Q. Vamos assumir que na execução do Algoritmo 4.1 sobre Q, os resultados das moedas lançadas são iguais aos resultados das moedas para as demandas na execução do algoritmo sobre Q'. Portanto, temos duas possibilidades para F:

(i) $F = F' \cup \{k\}$, com probabilidade $1/M$, ou

(ii) $F = F'$ com probabilidade $(1 - 1/M)$.

Se *(i)* k é incluído em F, então a função de cost-sharing com propriedade cross-monotonic do problema da árvore de Steiner implica que o custo compartilhado de cada demanda $j \in F'$ pode somente diminuir. Além disso, o custo de conexão de cada demanda $j \in Q' \setminus F'$ pode somente diminuir devido a opção adicional de se conectar a k.

Caso contrário, se *(ii)* k não é incluído em F, o custo compartilhado de cada facilidade $j \in F'$ permanece o mesmo, e o custo compartilhado de cada $j \in Q' \setminus F'$ pode somente diminuir, pois a distância do caminho mínimo de j para F somente diminui (através de $k \in Q' \setminus F'$).

Lema 7.1. *A função de cost-sharing ξ é cross-monotonic.*

Demonstração. Seja $Q' \subset D$ um subconjunto qualquer de demandas, e seja $Q = Q' \cup \{k\}$ para algum $k \notin Q'$. Além disso, F' e F são os conjuntos de facilidades abertas em Q' e Q respectivamente. Para completar a prova, é suficiente mostrar que para cada $j \in Q', \xi(Q', j) \geq \xi(Q, j)$. Temos que

$$\begin{aligned}
\mathrm{E}[\xi'(Q,j)] &= \sum_{O \subseteq Q} \mathrm{E}[\xi'(Q,j)|F = O] \cdot \Pr[F = O] \\
&= \sum_{O \subseteq Q'} \Big(\mathrm{E}[\xi'(Q,j)|F = O] \cdot \Pr[F = O] + \\
&\qquad \mathrm{E}[\xi'(Q,j)|F = O \cup \{k\}] \cdot \Pr[F = O \cup \{k\}] \Big).
\end{aligned}$$

Pelos argumentos apresentados acima, sabemos que para cada $j \in Q'$ e para cada $O \subseteq Q'$,

$$\begin{aligned}
\mathrm{E}[\xi'(Q,j)|F = O] &\leq \mathrm{E}[\xi'(Q',j)|F' = O], \quad \text{e} \\
\mathrm{E}[\xi'(Q,j)|F = O \cup \{k\}] &\leq \mathrm{E}[\xi'(Q',j)|F' = O].
\end{aligned}$$

Portanto,

$$\mathrm{E}[\xi'(Q,j)] \leq \sum_{O \subseteq Q'} \mathrm{E}[\xi'(Q',j)|F' = O] \cdot \Big(\Pr[F = O] + \Pr[F = O \cup \{k\}] \Big).$$

A prova agora segue do fato que, para cada $O \subseteq Q'$,

$$\Pr[F = O] + \Pr[F = O \cup \{k\}] = \Pr[F' = O].$$

Assim,

$$\begin{aligned}
\xi(Q,j) &= \frac{1}{4}\mathrm{E}[\xi'(Q,j)] \\
&\leq \frac{1}{4} \sum_{O \subseteq Q'} \mathrm{E}[\xi'(Q',j)|F' = O] \cdot \Pr[F' = O] \\
&= \frac{1}{4}\mathrm{E}[\xi'(Q',j)] \\
&= \xi(Q',j)
\end{aligned}$$

e o lema segue. ■

O próximo lema mostra que se $j \notin F$, então a função de cost-sharing de um usuário j pode ser calculada eficientemente, no entanto, no caso em que $j \in F$, não é conhecido como calcular eficientemente a esperança. Veremos mais adiante, no Capítulo 8, se utilizarmos uma marcação t-a-t independente (ver definição no Apêndice A) das demandas no Passo C1 do Algoritmo 4.1 ao invés da marcação totalmente independente (ver definição no Apêndice A) que utilizamos, então é possível uma desaleatorização deste.

Lema 7.2. *Seja $Q \subseteq D$, e seja $j \in D$ uma demanda. O custo de conexão esperado de j pode ser calculado em tempo polinomial.*

Demonstração. Considere o conjunto $Q^- = Q \setminus \{j\}$ de todas as demandas com excessão de j. Seja $v_1, v_2, \ldots, v_l$ o conjunto de demandas em Q^- ordenados de acordo com as distâncias não-decrescentes de j, onde $l = |Q^-|$. Ou seja, v_1 é a demanda mais próxima de j, v_l é a demanda mais distante de j. Então,

$$\begin{aligned}\mathrm{E}[\xi'(Q,j)|j \notin F] =& \frac{1}{M}c_{j,v_1} + \frac{1}{M}\left(1-\frac{1}{M}\right)c_{j,v_2} \\ &+ \frac{1}{M}\left(1-\frac{1}{M}\right)^2 c_{j,v_3} + \ldots + \frac{1}{M}\left(1-\frac{1}{M}\right)^{l-1} c_{j,v_l} \\ =& \frac{1}{M}\sum_{i=1}^{l}\left(1-\frac{1}{M}\right)^{i-1} c_{j,v_i}.\end{aligned}$$

Em outras palavras, j se conecta com alguma demanda v se v foi marcada (com probabilidade $1/M$) e nenhuma das i demandas anteriores foi marcada (com probabilidade $(1-1/M)^i$). ■

Para um subconjunto $Q \subseteq D$ de demandas, seja $\mathrm{val}(O, Q)$ a variável aleatória que denota o custo de uma solução O obtida pelo Algoritmo 4.1. O custo de uma solução ótima O^* para Q é denotado por $\mathrm{val}(O^*, Q)$.

Lema 7.3. *A função de cost-sharing $\xi(Q,j) = \frac{1}{4}E[\xi'(Q,j)]$ é competitiva, e para alguma constante $\varepsilon > 0$, com alta probabilidade, recupera pelo menos $\frac{1}{4}(1+\varepsilon)^{-1}$ da fração do custo de solução construída.*

Demonstração. Pelo Teorema 4.3, sabemos que o Algoritmo 4.1 é um algoritmo com fator de aproximação $(2 + \rho_{ST})$ para o Rent-or-Buy. Como estamos usando o algoritmo de Edmonds para aproximar a árvore geradora mínima, então $\rho_{ST} = 2$, e portanto o custo esperado de $\mathrm{E}[\mathrm{val}(O,Q)]$ é no máximo $4 \cdot \mathrm{val}(O^*, Q)$. Além disso, $\mathrm{E}[\mathrm{val}(O,Q)] = \sum_{j \in Q} \mathrm{E}[\xi'(Q,j)]$. Concluímos então que

$$\sum_{j \in Q} \xi(Q,j) = \frac{1}{4}\mathrm{E}\Big[\sum_{j \in Q}\xi'(Q,j)\Big] \leq \mathrm{val}(O^*, Q),$$

mostrando portanto que a função de custo-compartilhado $\xi(Q, j)$ é competitiva.

Pela desigualdade de Markov, com probabilidade no máximo $(1+\varepsilon)^{-1}$ e para qualquer constante $\varepsilon > 0$, temos que $\text{val}(O, Q) \geq (1+\varepsilon)\text{E}[\text{val}(O, Q)]$. Se executarmos várias vezes o algoritmo, podemos diminuir a probabilidade deste fornecer um valor maior que $(1+\varepsilon)\text{E}[\text{val}(O, Q)]$. Devemos então descobrir a quantidade k de vezes que devemos executar o algoritmo para obter baixas probabilidades de erro, i.e., probabilidade no máximo $\frac{1}{n}$ de que $\text{val}(O, Q) \geq (1+\varepsilon)\text{E}[\text{val}(O, Q)]$. Assim, devemos obter k tal que

$$\left(\frac{1}{1+\varepsilon}\right)^k = \frac{1}{n}.$$

Podemos reescrever a igualdade acima como

$$\begin{aligned}
&\log\left(\left(\frac{1}{1+\varepsilon}\right)^k\right) = \log\left(\frac{1}{n}\right) \\
\Rightarrow &\log\left(\frac{1}{(1+\varepsilon)^k}\right) = \log\left(\frac{1}{n}\right) \\
\Rightarrow &\log(1) - \log(1+\varepsilon)^k = \log(1) - \log(n) \\
\Rightarrow &\log(1+\varepsilon)^k = \log(n) \\
\Rightarrow &k \cdot \log(1+\varepsilon) = \log(n) \\
\Rightarrow &k = \frac{\log(n)}{\log(1+\varepsilon)}.
\end{aligned}$$

Portanto, executando o algoritmo $\log(n)/\log(1+\varepsilon)$ vezes, o Algoritmo 4.1 calcula uma solução tal que, com alta probabilidade (i.e., probabilidade pelo menos $1 - \frac{1}{n}$),

$$\text{val}(O, Q) \leq (1+\varepsilon)E[\text{val}(O, Q)].$$

Como $\text{E}[\text{val}(O, Q)] = \sum_{j \in Q} \text{E}[\xi'(Q, j)]$, então podemos escrever

$$\frac{\text{val}(O, Q)}{(1+\varepsilon)} \leq \sum_{j \in Q} \text{E}[\xi'(Q, j)].$$

Finalmente, concluímos que

$$\sum_{j \in Q} \xi(Q, j) = \frac{1}{4} \sum_{j \in Q} \text{E}[\xi'(Q, j)] \geq \frac{1}{4}(1+\varepsilon)^{-1} \cdot \text{val}(O, Q),$$

e o lema segue. ■

Combinando os Lemas 7.1 e 7.3, podemos enunciar o seguite teorema, resultado principal deste capítulo.

Teorema 7.4. *A função de cost-sharing $\xi(Q, j) = \frac{1}{4}E[\xi'(Q, j)]$ é cross-monotonic, e é budget-balance $\frac{1}{4}(1+\varepsilon)^{-1}$-aproximada.*

Capítulo 8

Técnica probabilística e sua desaleatorização

Neste capítulo, iremos obter uma função de cost-sharing para o problema Rent-or-Buy que é budget-balance β-aproximado, onde $\beta = 4.7$. Apesar do fator de aproximação da função de cost-sharing ser um pouco pior do que o fator obtido no Capítulo 7, iremos mostrar como os custos compartilhados podem ser calculados em tempo polinomial determinístico, e conseqüentemente, obtendo uma desaleatorização do algoritmo.

Usaremos o mesmo algoritmo utilizado no Capítulo 4 e a mesma função de cost-sharing proposta no Capítulo 7. No entanto, para calcular os custos esperados em tempo polinomial, iremos fazer uma análise diferente da análise que fizemos no Capítulo 4, permitindo a desaleatorização usando um espaço amostral pequeno (de tamanho polinomial no número de moedas lançadas), onde tal espaço contém algumas propriedades adicionais que ajudam a garantir a propriedade cross-monotonic.

Este capítulo está baseado nos resultados de Gupta, Srinivasan e Tardos em [13]. O resultado obtido por eles para o problema Rent-or-Buy é o melhor apresentado neste documento pois, além de desaleatorizar o Algoritmo 4.1 do Capítulo 4, é obtida uma função de cost-sharing para o problema, e sua razão de aproximação, de 4.7, é somente um pouco pior que o fator 4.55 obtido pelo Algoritmo 3.1 do Capítulo 3. Além disso, ainda em [13], os autores apresentam uma outra análise do algoritmo de forma a obter um fator de aproximação de 4.6. As idéias são bem parecidas com a que iremos apresentar neste capítulo e, portanto, esta análise será omitida na presente dissertação.

Toda nossa análise é para o problema Rent-or-Buy, mas iremos comentar no final do capítulo algumas modificações que devem ser feitas para tratar o Connected Facility Location.

Para este capítulo, iremos assumir como verdadeiras as Suposições 3.1 (demandas unitárias) e 3.2 (raiz na solução). Além disso, assumimos que há somente um jogador

em cada vértice. Iremos mostrar no final deste capítulo como descartar algumas dessas suposições.

8.1 Algoritmo

O algoritmo utilizado é igual ao proposto no Capítulo 4, com a única modificação que deixaremos livre a probabilidade da escolha do Passo C1. O algoritmo que utilizaremos é então descrito abaixo.

Algoritmo 8.1: RorB-Simples

Entrada: grafo $G = (V, E)$, demandas $D \subseteq V$, parâmetro $M > 1$;
Saída: árvore de Steiner T, facilidades F;

C1 Com probabilidade α/M, marque cada demanda $j \in D$.
Seja $D' \subseteq D$ o conjunto das demandas marcadas.

C2 Construa uma árvore de Steiner ρ_{ST}-aproximada em $F = D' \cup \{r\}$ e *compre* (i.e. pague custo $M \cdot c_e$) as arestas dessa árvore.
Os elementos de F são chamados *facilidades abertas*.

C3 Conecte cada demanda $j \in D$ à sua facilidade aberta mais próxima $i(j)$ em F.

devolva T, F

No algoritmo acima $\alpha > 0$ é uma constante que será escolhida mais tarde, e está baseada em nossa análise do algoritmo, como veremos mais a frente. A idéia é fazer a análise em cima da probabilidade α/M e no fim, fixar a constante α de tal forma que esta minimize o fator de aproximação do algoritmo.

8.2 Função de cost-sharing

O algoritmo sugere uma simples e intuitiva idéia para a função de cost-sharing: cada jogador paga um custo proporcional ao custo *esperado* da execução do algoritmo acima. Sabemos, pelo Capítulo 4, que o Algoritmo 8.1 é um algoritmo β-aproximado, para alguma constante β[1]. Para obter a função de cost-sharing com budget-balance β-aproximado, dividimos, para cada usuário, o seu custo esperado do Algoritmo 8.1 por β. Ou seja,

$$\xi(D, j) = \frac{1}{\beta} E[M\xi_{\text{AGM}}(F, j) + l(F, j)], \tag{8.1}$$

[1]No Capítulo 4 obtivemos um fator de aproximação β que será diferente do que vamos obter neste capítulo. Basta sabermos que existe uma constante β tal que o algoritmo é β-aproximado, onde a função de cost-sharing depende do valor de β.

onde $F = D' \cup \{r\}$, $l(S, j)$ é o caminho mínimo entre j e o vértice mais próximo de um conjunto S, e a esperança é sobre o conjunto F. Note que o conjunto F é probabilístico.

O resultado principal deste capítulo é descrito pelo seguinte teorema.

Teorema 8.1. *Há uma função de cost-sharing para o problema Rent-or-Buy que tem a propriedade cross-monotonic e é budget-balance β-aproximada (onde $\beta = 4.7$). Além disso, a função de cost-sharing pode ser calculada em tempo determinístico polinomial.*

Visão geral da prova

Vamos provar o Teorema 8.1 em duas partes: primeiramente temos que mostrar que as propriedades da função de cost-sharing são de fato satisfeitas, i.e., elas são budget-balance β-aproximada. Essas propriedades serão provadas na Seção 8.3.

Na segunda parte da prova, tecnicamente mais complicada, vamos mostrar que a função de cost-sharing pode ser calculada em tempo polinomial determinístico. Não é complicado ver que uma tentativa de desaleatorização do Algoritmo 8.1 se torna extremamente difícil, pois recai em esperanças condicionais difíceis de serem calculadas. Pela definição da esperança de uma variável aleatória, devemos calcular a probabilidade de todos os pontos do espaço probabilístico para obter o valor de uma esperança. O problema é que o espaço probabilístico do Algoritmo 8.1 tem tamanho exponencial no número de lançamentos de moedas. Para contornar esse problema, existem construções de espaços amostrais de tamanho polinomial que aproximam o espaço amostral original (de tamanho exponencial), no entanto, para realizar essas construções, devemos abandonar a *independência total* (ver definição no Apêndice A) nos lançamentos das moedas, e usar um lançamento em que as moedas sejam *t-a-t independentes* (ver definição no Apêndice A). Assim, devemos mostrar uma análise diferente do Algoritmo 8.1 para que esta aceite o fato das moedas serem t-a-t independentes. Deste modo, dadas uma marcação t-a-t independente das moedas no Passo C1 e uma análise do algoritmo proposto baseada nessa marcação, é possível construir um espaço amostral de tamanho polinomial, sendo possível então calcular a esperança das variáveis aleatórias em tempo polinomial e conseqüentemente, existe uma desaleatorização eficiente deste algoritmo. A construção de tal espaço é apresentada no Apêndice A. Esta nova análise do algoritmo será vista na Seção 8.4. Na Seção 8.5.2 veremos que se utilizarmos a marcação t-a-t independente, não irá modificar a aproximação obtida na Seção 8.4 (que foi analisado baseado na marcação totalmente independente).

Devemos, porém, ter certeza que a marcação limitada não vai influenciar nas propriedades provadas na Seção 8.3 (cross-monotonic e budget balance aproximado). Há vários métodos de construir variáveis aleatórias t-a-t independentes, no entanto, nem todas funcionam para que as propriedades da função de cost-sharing continuem valendo. Assim, na

prova do Teorema 8.8 mostramos que uma construção *particular* das variáveis aleatórias t-a-t independentes é crucial para que as propriedades continuem valendo.

8.3 Propriedades da função de cost-sharing

Nesta seção, iremos mostrar as propriedades da função de cost-sharing. Precisamos mostrar que a função definida na Equação (8.1) é uma função de cost-sharing com as propriedades budget-balance aproximada e cross-monotonic.

Para provar a propriedade de budget-balance aproximada, iremos precisar do seguinte resultado, que limita o desempenho do Algoritmo 8.1, cuja prova aparece na Seção 8.4.1.

Teorema 8.2. *Seja O^* uma solução ótima para o problema Rent-or-Buy. O custo esperado do Algoritmo 8.1 sobre um conjunto de demandas D é no máximo $\beta \cdot val(O^*, D)$ (onde $\beta = 4.7$), mesmo se as demandas são marcadas de modo t-a-t independentes no Passo (C1), para uma constante t suficientemente grande.*

O teorema acima, apesar de apresentar um fator de aproximação pior que o fator apresentado no Capítulo 4, é um resultado mais forte no sentido de que é necessário somente que as variáveis aleatórias sejam t-a-t independentes, onde t é uma constante, ao contrário do algoritmo anterior, onde deveríamos ter a independência total, n-a-n independente, onde n é uma variável (não-constante). Com este resultado, podemos agora provar as propriedades da função ξ e portanto provar o Teorema 8.1.

Antes, vamos estabelecer algumas das notações que iremos utilizar neste capítulo.

Notação 8.3. Denotamos por Y_j a variável aleatória que representa o fato da demanda j estar marcada, i.e., $Y_j = 1$ se a demanda j está marcada, $Y_j = 0$ caso contrário.

Notação 8.4. Seja ω uma amostra (i.e., uma seqüência de moedas lançadas pelo algoritmo) em nosso espaço amostral. Denotamos $Y_j(\omega)$ como segue: $Y_j(\omega) = 1$ se j é marcado pela sequência ω de arremessos de moedas, $Y_j(\omega) = 0$ caso contrário. Note que $Y_j(\omega)$ é definido deterministicamente, pois a amostra ω é dada.

Pelos Teoremas 5.12 e 5.13 e assumindo que o Teorema 8.2 é válido (veremos na Seção 8.4 a prova deste teorema), então podemos mostrar que, para qualquer amostra ω aleatória, que os custos-compartilhados ξ satisfazem a seguinte propriedade:

Lema 8.5. *A função ξ é budget-balance β-aproximada.*

Demonstração. Considere a função de cost-sharing $\beta \cdot \xi$. Vamos fixar uma amostra aleatória ω, e seja F_ω o conjunto das facilidades, i.e., $F_\omega = \{r\} \cup \{j \in D | Y_j(\omega) = 1\}$. Seja

val(AGM(F_ω)) o custo de uma árvore geradora mínima em F_ω. Da nossa definição de $\xi(D,i)$, temos:

$$\begin{aligned}\sum_{i\in D}\beta\xi(D,i) &= \mathrm{E}_\omega[M\sum_{i\in D}\xi_{\mathrm{AGM}}(F_\omega,i)+\sum_{i\in D}l(i,F_\omega)]\\ &= \mathrm{E}_\omega[M\cdot\mathrm{val}(\mathrm{AGM}(F_\omega))+\sum_{i\in D}l(i,F_\omega)].\end{aligned}\tag{8.2}$$

Lembrando que, pelos Teoremas 5.12 e 5.13, a função ξ_{AGM} é uma função de cost-sharing cross-monotonic, portanto $\xi_{\mathrm{AGM}}(F_\omega,i)=0$ para $i\notin F_\omega$ e $\sum_{i\in F_\omega}\xi_{\mathrm{AGM}}(F_\omega,i)=\mathrm{val}(\mathrm{AGM}(F_\omega))$.

Agora fica fácil verificar que a função de cost-sharing $\beta\cdot\xi$ recuperam o custo da solução construída, e portanto sua soma é pelo menos val(O,D), onde val(O,D) é o custo da solução O construída pelo algoritmo sobre o conjunto de demandas D. Isto é verdade, pois para cada elemento ω do espaço amostral, $M\cdot\mathrm{val}(\mathrm{AGM}(F_\omega))+\sum_{i\in D}l(i,F_\omega)$ é exatamente o custo da solução que abre facilidades em F_ω.

Seja O^* uma solução ótima para o problema. Para verificar que $\sum_{i\in D}\xi(D,i)\leq\mathrm{val}(O^*,D)$, note que o resultado final da Equação (8.2) é o custo esperado da execução do Algoritmo 8.1 no conjunto de demandas D, e portanto o Teorema 8.2 implica que o custo esperado não é maior que $\beta\cdot\mathrm{val}(O^*,D)$, e então $\sum_{i\in D}\xi(D,i)\leq\mathrm{val}(O^*,D)$. ■

8.3.1 Detalhes da marcação limitada

Antes de apresentar o tipo específico de marcação t-a-t independente que iremos usar, será útil ver como β depende do nosso parâmetro α. Primeiramente, iremos mostrar (na Seção 8.4.1) que, quando utilizamos a marcação totalmente independente, temos

$$\beta\leq\max\left(2(1+\alpha),\ 2+\frac{2e^\alpha}{e^\alpha-1}\right).\tag{8.3}$$

Mostraremos então (na Seção 8.5.2) que este limitante muda muito pouco se t é uma constante suficientemente grande: se $t=a\log(1/\epsilon)$, onde a é uma constante, então o lado direito da desigualdade acima é multiplicado por no máximo $(1+\epsilon)$. Portanto, fazendo $\alpha\approx 1.35$, garantimos que $\beta\leq 4.7$.

Devemos mostrar que a esperança usando a marcação t-a-t independente tem a propriedade de ser cross-monotonic. Para isso, devemos fazer uma marcação específica. Como dito anteriormente, há vários modos de realizar a marcação de independência limitada, no entanto, a marcação específica que iremos apresentar nos leva à obter a propriedade cross-monotonic. Iremos descrever como é feita tal marcação a seguir.

Seja $\mathcal{F}$ um *corpo finito* (ver definição no Apêndice A) de tamanho pelo menos $n=|V|$, onde $|\mathcal{F}|$ é grande suficiente para que $\lceil\alpha|\mathcal{F}|/M\rceil/|\mathcal{F}|$ seja suficientemente próximo de α/M.

Em particular, vamos supor que $|\mathcal{F}| \geq \Omega(M \log n)$. Sejam $\{a_1, a_2, \ldots, a_{|\mathcal{F}|}\}$ os elementos do corpo $\mathcal{F}$, sendo que os vértices V são dados pelos n primeiros elementos do corpo $\{a_1, a_2, \ldots, a_n\}$. Seja $\mathcal{S}$ algum subconjunto pré-especificado de $\mathcal{F}$, tal que

$$|\mathcal{S}| = \left\lceil \frac{\alpha}{M} \cdot |\mathcal{F}| \right\rceil.$$

Seja $\omega = (x_0, x_1, \ldots, x_{t-1})$ um vetor de tamanho t que foi obtido de $\mathcal{F}^t = \mathcal{F} \times \mathcal{F} \times \ldots \times \mathcal{F}$ aleatoriamente e com distribuição uniforme.

Para obtermos a amostra $Y = \{Y_1, \ldots, Y_n\}$, definimos Y_i igual a 1 se o valor da soma $\sum_{j=0}^{t-1} x_j a_i^j$ pertencer a $\mathcal{S}$, e $Y_i = 0$ caso contrário. Lembrando que Y_i é a variável aleatória que indica se a demanda i foi marcada, e portanto devemos mostrar que, utilizando o modo de marcação descrito na frase anterior, temos $\Pr[Y_i = 1] = \alpha/M$. Isso é facilmente observado, pois $\mathcal{F}$ sendo um corpo finito, qualquer seqüência de operações de soma e multiplicação neste corpo (em particular, a seqüência de operações $\sum_{j=0}^{t-1} x_j a_i^j$) tem como resultado um elemento do próprio corpo. Em outras palavras,

$$\Pr\left[\sum_{j=0}^{t-1} x_j a_i^j \text{ pertencer a } \mathcal{F}\right] = 1.$$

Como $\mathcal{S}$ é subconjunto aleatório de $\mathcal{F}$, e tem aproximadamente α/M do tamanho de $\mathcal{F}$, então $\Pr[Y_i = 1]$ é igual a

$$\Pr\left[\sum_{j=0}^{t-1} x_j a_i^j \text{pertencer a } \mathcal{S}\right] \approx \frac{\alpha}{M}.$$

De maneira mais formal, como $\alpha|\mathcal{F}|/M \leq |\mathcal{S}| \leq \alpha|\mathcal{F}|/M + 1$ (pela definição de $|\mathcal{S}|$) então

$$\frac{\alpha}{M} \leq \Pr[Y_i = 1] \leq \frac{\alpha}{M} + \frac{1}{|\mathcal{F}|}.$$

Em particular, como $|\mathcal{F}| \geq \Omega(M \log n)$ então

$$\frac{\alpha}{M} \leq \Pr[Y_i = 1] \leq \frac{\alpha}{M} + \frac{1}{|\mathcal{F}|} = \frac{\alpha + o(1)}{M};$$

e pela Equação (8.3) e as frases seguintes a ela, vemos que a troca de α por $\alpha + o(1)$ muda β somente de um fator $o(1)$ adicional. Portanto, em todos nossos argumentos futuros, iremos assumir que $\Pr[Y_i = 1] = \alpha/M$ para todo $i \in D$. Nos resta agora mostrar que essa construção nos fornece a marcação das variáveis com dependência limitada. De fato, na construção apresentada, as variáveis são todas t-a-t independentes, e a prova disto é mostrada no Apêndice A.

Note que a distribuição acima está gerando n lançamentos de moedas Y_i, mas precisamos de somente $|D|$ lançamentos, para isto, basta ignorar todos Y_j para $j \notin D$.

Notação 8.6. Para um $\omega \in \mathcal{F}^t$ fixo, $F(\omega)$ denota o conjunto das facilidades, i.e., $F(\omega) = \{r\} \cup \{j \in D | Y_j(\omega) = 1\}$.

Note que $F(\omega)$ tem seu resultado de forma determinística, pois ω é dado e $Y_j(\omega)$ também é determinístico, como observado na Notação 8.3.

Lema 8.7. *Sejam A, D conjuntos de demandas, com $A \subseteq D$. Seja ω uma amostra do espaço probabilístico $\mathcal{F}^t$. A marcação t-a-t independente que utilizamos tem a seguinte propriedade crucial: para qualquer ω, se o conjunto das demandas marcadas numa execução do Algoritmo 8.1 em A é $A'(\omega)$ e o conjunto das demandas marcadas numa execução do Algoritmo 8.1 em D é $D'(\omega)$, então $A'(\omega) \subseteq D'(\omega)$.*

Demonstração. Como ω é fixo e igual para as duas execuções, os $Y_j(\omega)$ são funções *determinísticas* idênticas para as duas execuções, ou seja, as mesmas demandas são marcadas, independentemente do conjunto que estamos trabalhando. Mas como $A \subseteq D$, claramente $A'(\omega) \subseteq D'(\omega)$ e a afirmação segue. ■

Teorema 8.8. *Assumindo que a marcação aleatória do Passo (C1) é feita utilizando variáveis aleatórias t-a-t independentes, ou usando variáveis aleatórias (totalmente) independentes, então a função ξ é cross-monotonic, i.e., $\xi(D, i) \leq \xi(A, i)$ para qualquer $A \subseteq D$.*

Demonstração. Definimos a probabilidade $p(F, E)$ como sendo a probabilidade de, ao selecionar e fixar uma amostra ω do espaço amostral, o conjunto das demandas marcadas numa execução do Algoritmo 8.1 sobre A é F, e o conjunto das demandas marcadas numa execução do Algoritmo 8.1 sobre D é E. Note que se $p(F, E) > 0$, então obrigatoriamente $F \subseteq E$, pois como afirmamos no Lema 8.7, o caso em que $F \supset E$ nunca ocorre e como conseqüência, para $F \supset E$, a probabilidade $p(F, E) = 0$.

Analogamente, de forma mais simples, podemos notar que o fato anterior e o Lema 8.7 também valem para o caso em que a marcação é feita de forma totalmente independente. Com base nisto, podemos provar que ξ é cross-monotonic, da seguinte forma.

$$\begin{aligned}
\xi(A, i) &= \frac{1}{\beta}\mathrm{E}[M\xi_{\mathrm{AGM}}(F, i) + l(i, F)] \\
&= \frac{1}{\beta}\textstyle\sum_{F,E} p(F, E)[M\xi_{\mathrm{AGM}}(F, i) + l(i, F)] \\
&\geq \frac{1}{\beta}\textstyle\sum_{F,E} p(F, E)[M\xi_{\mathrm{AGM}}(E, i) + l(i, E)] \\
&= \frac{1}{\beta}\mathrm{E}[M\xi_{\mathrm{AGM}}(E, i) + l(i, E)] \\
&= \xi(D, i)
\end{aligned}$$

Iremos explicar agora a razão da desigualdade acima ser válida. Sabemos que se $p(F, E) > 0$ então $F \subseteq E$, portanto os únicos conjuntos levados em consideração no somatório são

os conjunto em que $F \subseteq E$. Pelo Teorema 5.13, sabemos que ξ_{AGM} é cross-monotonic, portanto $\xi_{\text{AGM}}(F, i) \geq \xi_{\text{AGM}}(E, i)$ para $F \subseteq E$. Se $F \subseteq E$, então a distância de i até F só pode ser maior ou igual a distância de i até E, pois E contém todos os elementos de F e elementos adicionais que não pertencem a F podem estar mais perto de i. Em outras palavras $l(i, F) \geq l(i, E)$. Assim, a desigualdade anterior é válida. ■

8.4 Uma nova análise do algoritmo

Nesta seção, vamos fazer uma nova análise do Algoritmo 8.1. Vamos supor, para esta seção, que a marcação é feita de maneira *totalmente independente* no Passo (C1) do algoritmo. Iremos então usar esta nova análise na Seção 8.5 para mostrar que nosso valor de β obtido tem uma mudança insignificante quando utilizarmos a marcação t-a-t indepedente das variáveis aleatórias, para um valor grande suficiente (mas ainda sendo uma constante) de t.

Seja O^* uma solução ótima para o Rent-or-Buy, onde F^* é o conjunto de facilidades da solução O^* e T^* é a árvore de Steiner sobre F^* na solução ótima O^*. Seja $\text{ccon}(O^*)$ e $\text{cste}(O^*) = M \cdot \text{val}(T^*)$ o custo de conexão e o custo da árvore de Steiner respectivamente na solução ótima O^*. Além disso, definimos $\text{val}(O^*) = \text{ccon}(O^*) + \text{cste}(O^*)$.

O prova do limitante superior do custo da árvore de Steiner construída no Passo (C2) é igual à prova do limitante do Lema 4.1 obtido no Capítulo 4: consideramos a solução ótima O^*, e calculamos o custo esperado assumindo que iremos construir a árvore de Steiner do Passo (C2) usando os caminhos da solução ótima. No entanto, o limitante é em função da constante α, tornando-se um pouco diferente do resultado já apresentado. Vamos mostrar a prova completa deste lema a seguir. A parte (a) do lema somente será utilizada na Seção 8.5. As variáveis aleatórias Y_j são definidas como na Notação 8.3.

Lema 8.9. *(a) O custo esperado S do Passo (C2) do Algoritmo 8.1 é limitado por uma combinação linear não negativa dos valores em $\{E[Y_j] : j \in D\}$; (b) esta combinação linear não negativa é no máximo*

$$2 \cdot (cste(O^*) + \alpha\, ccon(O^*)).$$

Demonstração. Considere uma solução ótima O^*: ela conecta cada $j \in D$ a uma facilidade $i^*(j) \in F^*$ através do caminho mais curto, e portanto paga um custo de conexão $\text{ccon}(O^*) = \sum_{j \in D} c_{j,i^*(j)}$. Seja F como definido no Passo (C2) do algoritmo. Iremos definir uma árvore que conecta F da seguinte forma: compre todas as arestas em T^*, formando a árvore ótima, e para cada $j \in F \setminus F^*$, compre as arestas do menor caminho de j até $i^*(j)$ usado na solução ótima, aumentando a árvore. Note que pagamos $M \cdot c_e$ para cada aresta e no Passo (C2), e portanto pagamos $M \cdot \text{val}(T^*) = \text{cste}(O^*)$ pelas arestas da

árvore ótima T^*. Para os caminhos aleatórios que adicionamos à árvore, o custo esperado do caminho entre algum j e $i^*(j)$ é $\mathrm{E}[Y_j] \cdot M \cdot c_{j,i^*(j)}$. Portanto, o custo esperado total desta árvore que conecta F é no máximo

$$\mathrm{cste}(O^*) + \sum_{j \in D} \mathrm{E}[Y_j] \cdot M \cdot c_{j,i^*(j)}.$$

Como usamos uma árvore geradora mínima no Passo (C2), pelo Teorema 2.10 do Capítulo 2, o custo esperado de comprar a árvore T é no máximo duas vezes a equação acima, ou seja, no máximo

$$2 \cdot \left(\mathrm{cste}(O^*) + \sum_{j \in D} \mathrm{E}[Y_j] \cdot M \cdot c_{j,i^*(j)} \right),$$

provando então a parte (a) do lema. Para provar a parte (b) do lema, basta verificar que $\mathrm{E}[Y_j] = \alpha/M$, e então o custo da árvore fica no máximo

$$\begin{aligned} & 2 \cdot \left(\mathrm{cste}(O^*) + \sum_{j \in D} \frac{\alpha}{M} \cdot M \cdot c_{j,i^*(j)} \right) \\ &= 2 \cdot \left(\mathrm{cste}(O^*) + \alpha \sum_{j \in D} c_{j,i^*(j)} \right) \\ &= 2 \cdot (\mathrm{cste}(O^*) + \alpha \mathrm{ccon}(O^*)). \end{aligned}$$

■

Modificando a solução ótima

Antes de limitar o custo de conexão, vamos modificar a solução ótima O^* para obter outra solução que chamaremos de O'^*. Esta nova solução tem custo maior que O, mas sua estrutura nos permite limitar o custo de conexão e provar as propriedades da marcação t-a-t independente que serão vistas na Seção 8.5.

Lema 8.10. *Há uma solução O'^* com as seguintes propriedades:*

P1 *O'^* abre facilidades em $F' \subseteq F^*$, e estas são conectadas por um* ciclo *T' ao invés da árvore T^*; cada demanda j é agora conectada à facilidade $i'(j)$, que não é necessariamente sua facilidade mais próxima.*

P2 *O número de demandas atribuídas a uma facilidade em F', exceto talvez na raiz r, é um múltiplo de M. Iremos chamar essa propriedade de restrição* mod-M.

Se O'^* tem as propriedades acima, então O'^* tem custo de conexão $ccon(O'^*) \leq ccon(O^*) + cste(O^*)$ e custo de Steiner $cste(O'^*) \leq 2 \cdot cste(O^*)$. Portanto o custo total de O'^* é $val(O'^*) \leq ccon(O^*) + 3 \cdot cste(O^*)$.

Demonstração. A seguir, iremos modificar a atribuição das demandas aos vértices em F^* para satisfazer a restrição mod-M. Dada a árvore T^*, iremos fazer isso em dois passos.

O primeiro passo vai de baixo para cima, nível a nível da árvore, começando no nível mais profundo com relação a r, que contém folhas, que devem estar todas em F^*, i.e., nenhuma folha é um vértice não-terminal, lembrando que uma árvore de Steiner é formada por vértices terminais e possivelmente vértices não-terminais. Vale observar que nem todos vértices em F^* são folhas que estão no nível mais profundo, podem existir folhas que estão em níveis mais altos (ver Fig. 8.1). Considere um vértice em que todos os descendentes satisfazem a restrição mod-M (no caso das folhas, estas não têm descendentes), e seja x a demanda atribuída a ele. Se $x = aM + b$, onde $b < M$, enviamos b unidades de demanda de x para seu pai. Fazendo isso para todos os vértices da árvore, o número de demandas atribuídas a cada vértice é agora um múltiplo de M, com excessão talvez da raiz r, que não pode enviar seu "excesso" b para nenhum outro vértice.

A propriedade P1 diz que abriremos um conjunto $F' \subseteq F^*$ de facilidades. No entanto, após o primeiro passo, que vai de baixo para cima, possivelmente alguns vértices não-terminais, i.e., vértices que pertencem à árvore mas não pertencem a F^*, podem estar com pelo menos M demandas atribuídas, e conseqüentemente deveríamos abrir facilidades neste vértices. Mas como queremos respeitar a propriedade P1, não podemos abrir facilidade nesses vértices não-terminais. Assim, no segundo passo, que é feito de cima para baixo na árvore, enviamos demandas para baixo na árvore. Seja um vértice $v \notin F^*$ que pertence à árvore e que tem demanda aM (vamos assumir que todos os seus ancestrais já foram tratados). Como pelo menos $(a-1)M + 1$ desta demanda foi coletada no primeiro passo (pois inicialmente v tinha demanda menor que M), e cada filho de v só pode ter enviado menos que M demandas a ele, então deve haver pelo menos a filhos para v enviar demandas para baixo. Assim, escolhemos a filhos quaisquer e enviamos a cada um deles M demandas, fazendo com que a propriedade P2 ainda continue satisfeita. Observe que v não tem mais demandas atribuídas. O processo então continua para seus descendentes, de cima para baixo.

Para completar a construção de O'^*, note que o fluxo de demandas enviadas ao longo de uma aresta de T^* foi de b unidades para cima na árvore ou $M - b$ unidades para baixo, e então o custo deste movimento pode ser trocado por $\text{cste}(O^*)$, portanto temos que $\text{ccon}(O'^*) \leq \text{ccon}(O^*) + \text{cste}(O'^*)$. Além disso, como no final não usamos nenhuma facilidade adicional e ainda podemos fechar as facilidades em que não há demandas atribuídas, então $F' \subseteq F^*$. Finalmente, a propriedade do ciclo é obtida da seguinte forma. As arestas

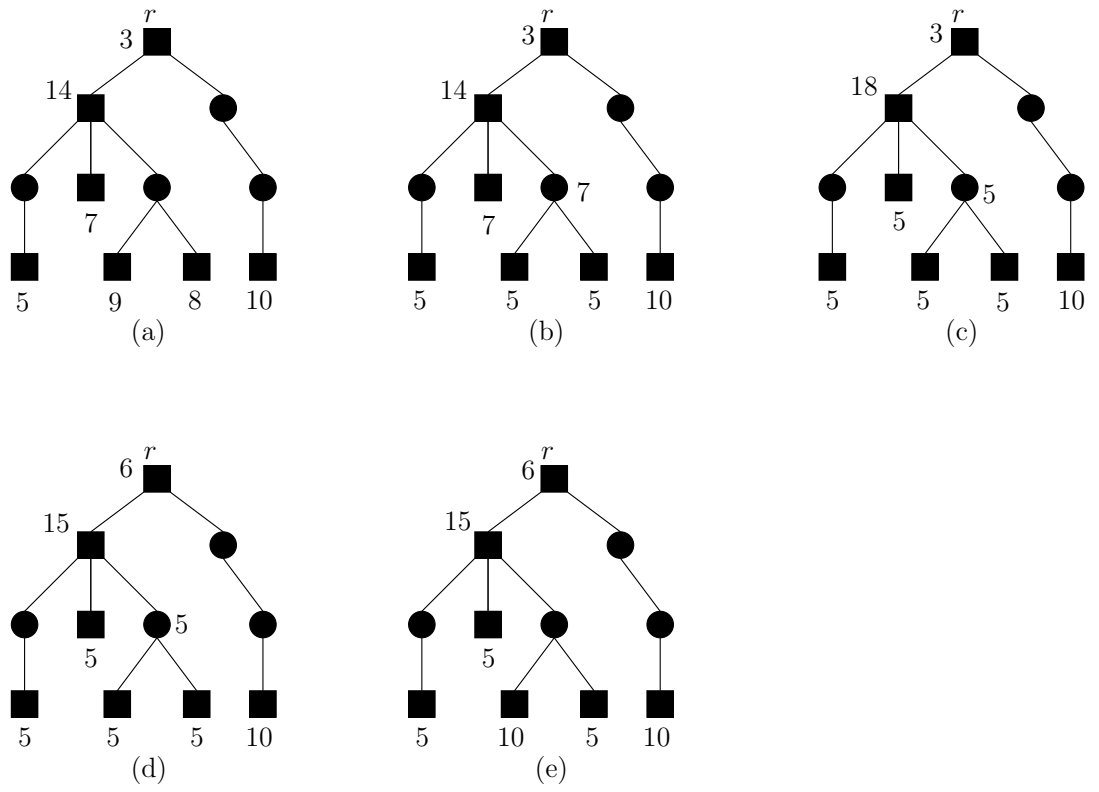

Figura 8.1: Supondo $M = 5$. (a) Estado inicial, onde os quadrados representam as facilidades abertas, os círculos os vértices não terminais da árvore de Steiner, e os números são os pesos de demandas atribuídos a cada facilidade aberta. (b) Primeiro passo, que vai de baixo para cima, começa no nível mais abaixo da árvore, enviando demandas para um nível acima para tornar os pesos deste nível múltiplos de M. (c)Continuação do primeiro passo, agora agindo um nível acima na árvore. (d) Continuação do primeiro passo. Agora foram processados todos os níveis da árvore, e todos os vértices (exceto a raiz) contém pesos de demandas que são múltiplos de M. (e)Segundo passo, que vai de cima para baixo. Neste passo, tratamos todos os vértices não terminais que contêm peso de demanda > 0. No caso deste exemplo, há somente um vértice neste caso, então enviamos sua demanda para algum vértice filho.

da árvore T^* são duplicadas (gerando arestas paralelas) e com isso é possível obter um circuito Euleriano de T^*. Tal circuito pode ser substituído por outro circuito que não repete vértices e passa por vértices de F^* sem aumentar seu custo. Fazer isto na pior das hipóteses duplica o custo da árvore de Steiner, obtendo $\text{cste}(O'^*) \leq 2\text{ccon}(O^*)$. ■

De agora em diante, vamos considerar somente a solução modificada O'^*. Por questões de simplicidade, vamos assumir que cada vértice em F' tem M demandas atribuídas. Isso

pode ser garantido da seguinte forma: se $f \in F'$ tem demanda aM, então fazemos a cópias idênticas de f. Lembrando que T' é um ciclo que contém r, vamos nomear as facilidades em T', começando em r, em (digamos) sentido horário $r = f_0, f_1, \ldots, f_k, f_{k+1} = r$.

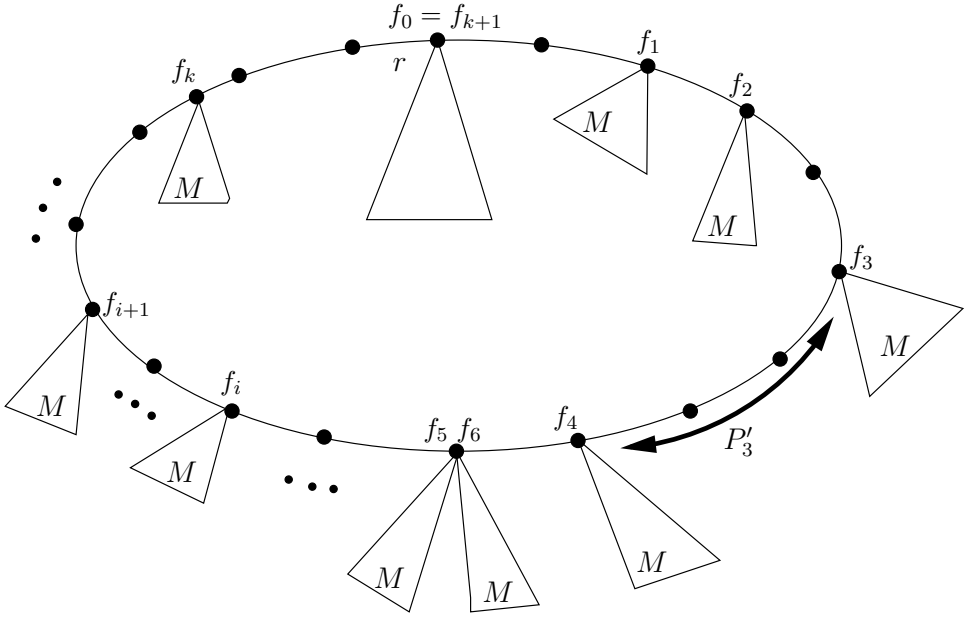

Figura 8.2: A instância transformada O'^*.

Seja D_l o conjunto de demandas atribuídas a f_l, e portanto $|D_l| = M$ para $l \neq 0$ pela propriedade P2 do Lema 8.10. Vamos abusar de notação e adicionar r a D_0. Seja P'_l o caminho do ciclo T' que liga f_l a f_{l+1} (ver Fig 8.2), e seja $c(P'_l)$ o comprimento de P'_l. Portanto $\text{cste}(O'^*) = \sum_{l=0}^{k} c(P'_l)$. Seja C'_l o custo total de conexão das demandas em D_l na solução O'^*. Portanto $\text{ccon}(O'^*) = \sum_{l=0}^{k} C'_l$. No Passo (C3), o Algoritmo 8.1 escolhe a atribuição mais barata das demandas aos vértices em F. Portanto, para limitar o custo de conexão esperado, basta limitar o custo de conexão esperado de um experimento qualquer que atribui demandas aos vértices em F. Iremos apresentar e analisar um possível experimento na próxima seção, sendo que em [13] um segundo experimento mais elaborado é analisado, obtendo assim um fator de aproximação de 4.6. No experimento a seguir, obtemos um fator de aproximação de 4.7.

8.4.1 Uma possível atribuição

Iremos considerar o seguinte experimento: se $D_l \cap F \neq \emptyset$, então atribuímos *todas* as demandas em D_l ao elemento em $D_l \cap F$ que está mais próximo a f_l. Caso contrário, temos que atribuir essas demandas a alguma outra facilidade aberta (i.e., "ir para fora

de D_l"). Neste caso, considere o menor $t > 0$ tal que $D_{l+t} \cap F \neq \emptyset$, seja $s = l + t$. Note que $r \in D_0$ e portanto $t \leq k$. Enviamos então todas as demandas em D_l para a facilidade em $D_s \cap F$ que está mais perto de f_s. Se atribuirmos as demandas em D_l a um vértice marcado em D_{l+t} então um caminho para cada demanda em D_l passa por $f_l, P'_l, P'_{l+1}, \ldots, P'_{l+t-1}$ e então de f_{l+t} ao elemento de $D_{l+t} \cap F$ mais perto de f_{l+t}. Iremos limitar o custo esperado destes caminhos, o qual por sua vez irá limitar o custo de conexão esperado do Algoritmo 8.1.

Vamos definir algumas variáveis aleatórias. Seja X_i a variável que indica se $D_i \cap F = \emptyset$ em nosso algoritmo, ou seja $X_i = 0$ se $D_i \cap F \neq \emptyset$ e $X_i = 1$ se $D_i \cap F = \emptyset$. Note que $X_0 = 0$ com probabilidade 1, pois $r \in D_0$. Seja A_i a variável aleatória da distância de f_i ao elemento mais próximo de $D_i \cap F$, se $D_i \cap F$ é vazio, então $A_i = 0$. Pelos argumentos anteriores, o custo de atribuição das M demandas em D_l é no máximo

$$C'_l + M \sum_{i=l}^{k} (X_l \ldots X_i) c(P'_i) + M \sum_{i=l}^{k} (X_l \ldots X_{i-1})(1 - X_i) A_i. \tag{8.4}$$

O primeiro termo da equação acima é a distância que as demandas em D_l devem caminhar para alcançar f_l. O segundo termo diz respeito às demandas em D_l que usam P'_i (pagando $M \cdot c(P'_i)$). Essas demandas usam P'_i se e somente se nenhum $D_l, \ldots, D_i$ tem intersecção com F, ou em outras palavras, se $X_l = 1, \ldots, X_i = 1$ então as demandas em D_l utilizam o caminho P'_i. O terceiro termo diz respeito a pagar um custo de $M \cdot A_i$ se atribuirmos demandas em D_l ao elemento mais próximo de $D_i \cap F$, ou seja, os elementos de D_l pagarão $M \cdot A_i$ se e somente se $X_l = 1, X_{l+1} = 1, \ldots, X_{i-1} = 1$ e $X_i = 0$.

Note que, para $i \neq j$, como D_i e D_j são disjuntos, então X_i e X_j são independentes. A probabilidade de um elemento não ser marcado é $(1 - \alpha/M)$, a probabilidade de nenhum dos M elementos de D_l ter sido marcado é $(1 - \alpha/M) \cdot (1 - \alpha/M) \cdot \ldots \cdot (1 - \alpha/M) = (1 - \alpha/M)^M$. Portanto

$$\begin{aligned} \mathrm{E}[X_i] &= 0 \cdot \Pr(D_l \cap F \neq \emptyset) + 1 \cdot \Pr(D_l \cap F = \emptyset) \\ &= 0 + (1 - \alpha/M)^M \\ &= (1 - \alpha/M)^M. \end{aligned}$$

Para limitar $M \cdot \mathrm{E}[(1 - X_i)A_i]$, seja $a_1, \ldots, a_M$ as distâncias de f_i aos elementos de D_i, onde

$$0 \leq a_1 \leq a_2 \leq \ldots \leq a_M, \text{com} \sum_{j \in D_i} a_j = C'_i. \tag{8.5}$$

Em outras palavras, as distâncias a_i são uma ordenação, onde a_1 é o elemento de D_i que está mais próximo de f_i e a_M é o elemento de D_i que está mais distante de f_i. Assim

$$\mathrm{E}[(1-X_i)A_i] = \sum_{j=1}^{M} a_j \cdot \alpha/M \cdot (1-\alpha/M)^{j-1}, \tag{8.6}$$

ou seja, a_j é marcado com probabilidade α/M e a probabilidade de nenhum dos a's anteriores terem sido marcados é $(1-\alpha/M)^{j-1}$. A medida que j aumenta, os coeficientes $\alpha/M \cdot (1-\alpha/M)^{j-1}$ diminuem, portanto quando sujeito às restrições (8.5), a esperança tem seu maior valor quando todos a_j's são iguais a C'_i/M. Portanto

$$\begin{aligned}
\mathrm{E}[(1-X_i)A_i] &\leq \sum_{j=1}^{M} \frac{C'_i}{M} \cdot \frac{\alpha}{M} \cdot (1-\frac{\alpha}{M})^{j-1} \\
&= \frac{C'_i}{M} \cdot \frac{\alpha}{M} \cdot \sum_{j=1}^{M} (1-\frac{\alpha}{M})^{j-1} \\
&= \frac{C'_i}{M} \cdot \frac{\alpha}{M} \cdot \sum_{j=0}^{M-1} (1-\frac{\alpha}{M})^{j}.
\end{aligned}$$

Por questões de clareza nas equações, vamos chamar $\mathrm{E}[X_i] = (1-\alpha/M)^M$ de q. Assim, usando a fórmula de somatório de progressão geométrica, temos

$$\begin{aligned}
\frac{C'_i}{M} \cdot \frac{\alpha}{M} \cdot \sum_{j=0}^{M-1} (1-\frac{\alpha}{M})^{j} &= \frac{C'_i}{M} \cdot \frac{\alpha}{M} \cdot \left(\frac{1-\left(1-\frac{\alpha}{M}\right)^M}{1-\left(1-\frac{\alpha}{M}\right)} \right) \\
&= \frac{C'_i}{M} \cdot \frac{\alpha}{M} \cdot \left(\frac{1-\left(1-\frac{\alpha}{M}\right)^M}{\frac{\alpha}{M}} \right) \\
&= \frac{C'_i}{M} \cdot \left(1-\left(1-\frac{\alpha}{M}\right)^M\right) \\
&= \frac{C'_i}{M} \cdot (1-q).
\end{aligned}$$

Portanto $\mathrm{E}[(1-X_i)A_i] \leq C'_i/M \cdot (1-q)$.

Seja C_l o custo esperado de conexão das demandas em D_l em nosso atual experimento. Assim, pela Desigualdade (8.4), temos que, para $l > 0$,

$$C_l = \mathrm{E}\left[C'_l + M \sum_{i=l}^{k} (X_l \dots X_i) c(P'_i) + M \sum_{i=l}^{k} (X_l \dots X_{i-1})(1-X_i)A_i \right],$$

que pela linearidade da esperança nos dá

$$C_l = C'_l + M \sum_{i=l}^{k} \mathrm{E}\left[(X_l \dots X_i)\right] c(P'_i) + M \sum_{i=l}^{k} \mathrm{E}\left[(X_l \dots X_{i-1})(1-X_i)A_i\right].$$

Além disso, como afirmado anteriormente, as variáveis aleatórias X_i são independentes, portanto

$$C_l = C'_l + M\sum_{i=l}^{k}(\mathrm{E}[X_l]\dots\mathrm{E}[X_i])c(P'_i) + M\sum_{i=l}^{k}(\mathrm{E}[X_l]\dots\mathrm{E}[X_{i-1}])\mathrm{E}[(1-X_i)A_i].$$

Combinando isso com os fatos vistos acima, lembrando que $q = (1-\alpha/M)^M = \mathrm{E}[X_i]$, temos

$$C_l \leq C'_l + M\sum_{i=l}^{k} c(P'_i)q^{i-l+1} + \sum_{i=l}^{k} C'_i(1-q)q^{i-l}.$$

Como $C_0 = C'_0$, e somando a desigualdade acima sobre todos os l, temos que o custo total esperado de uma solução O obtida é

$$\mathrm{ccon}(O) \leq \sum_{l=0}^{k} C'_l + M\sum_{l=1}^{k}\sum_{i=l}^{k} c(P'_i)q^{i-l+1} + \sum_{l=1}^{k}\sum_{i=l}^{k} C'_i(1-q)q^{i-l}. \tag{8.7}$$

Podemos modificar o somatório duplo, basta atentar ao fato que este varia de $[1 \leq l \leq k][l \leq i \leq k]$, e pode ser reescrito como um único somatório que varia de $[1 \leq l \leq i \leq k]$, que pode novamente ser reescrito como somatório duplo que varia de $[1 \leq i \leq k][1 \leq l \leq i]$. Ou seja

$$[1 \leq l \leq k][l \leq i \leq k] \equiv [1 \leq l \leq i \leq k] \equiv [1 \leq i \leq k][1 \leq l \leq i].$$

Assim, podemos reescrever a Desigualdade (8.7) da seguinte forma

$$\mathrm{ccon}(O) \leq \sum_{l=0}^{k} C'_l + M\sum_{i=1}^{k}\sum_{l=1}^{i} c(P'_i)q^{i-l+1} + \sum_{i=1}^{k}\sum_{l=1}^{i} C'_i(1-q)q^{i-l},$$

que com um pouco mais de manipulação algébrica e utilizando a fórmula de somatório de progressão geométrica, obtemos

$$\begin{aligned}
\mathrm{ccon}(O) \leq & \sum_{l=0}^{k} C'_l + M\sum_{i=1}^{k} c(P'_i)\sum_{l=1}^{i} q^{i-l+1} + \sum_{i=1}^{k} C'_i(1-q)\sum_{l=1}^{i} q^{i-l} \\
\leq & \mathrm{ccon}(O'^*) + M\sum_{i=1}^{k} c(P'_i)\frac{q}{1-q} + \sum_{i=1}^{k} C'_i(1-q)\frac{1}{1-q} \\
= & \mathrm{ccon}(O'^*) + M\sum_{i=0}^{k} c(P'_i)\frac{q}{1-q} + \sum_{i=0}^{k} C'_i \\
= & \mathrm{ccon}(O'^*) + \frac{q}{1-q}\mathrm{cste}(O'^*) + \mathrm{ccon}(O'^*) \\
= & 2\cdot\mathrm{ccon}(O'^*) + \frac{q}{1-q}\mathrm{cste}(O'^*)
\end{aligned}$$

Sabemos, pela definição de e (constante de Euler), que

$$\lim_{M\to\infty}\left(1-\frac{1}{M}\right)^M=\frac{1}{e}.$$

Modificando um pouco, temos

$$\begin{aligned}
&\lim_{M\to\infty}\left(1-\frac{\alpha}{\alpha\cdot M}\right)^M=\frac{1}{e}\\
\Longrightarrow&\lim_{M\to\infty}\left(\left(1-\frac{\alpha}{\alpha\cdot M}\right)^M\right)^\alpha=\left(\frac{1}{e}\right)^\alpha\\
\Longrightarrow&\lim_{M\to\infty}\left(1-\frac{\alpha}{\alpha\cdot M}\right)^{\alpha\cdot M}=\frac{1}{e^\alpha}\\
\Longrightarrow&\lim_{M\to\infty}\left(1-\frac{\alpha}{M'}\right)^{M'}=\frac{1}{e^\alpha}.
\end{aligned}$$

Disso, como M não tende ao infinito, temos que, $q\leq 1/e^\alpha$. Portanto, o segundo termo da inequação anterior é no máximo $1/(e^\alpha-1)$. Agora, usando o Lema 8.10 para substituir $\mathrm{ccon}(O'^*)$ por $\mathrm{ccon}(O^*)+\mathrm{cste}(O^*)$ e $\mathrm{cste}(O'^*)$ por $2\cdot\mathrm{ccon}(O^*)$, temos que o custo de conexão esperado do nosso algoritmo pode ser limitado por

$$\mathrm{ccon}(O)\leq 2\cdot\mathrm{ccon}(O^*)+\frac{2e^\alpha}{e^\alpha-1}\cdot\mathrm{cste}(O^*).$$

Somando isso ao Lema 8.9, o custo total esperado é no máximo

$$\mathrm{ccon}(O)+\mathrm{cste}(O)\leq 2(1+\alpha)\mathrm{ccon}(O^*)+(2+(2e^\alpha)/(e^\alpha-1))\mathrm{cste}(O^*). \tag{8.8}$$

Como o custo ótimo é $\mathrm{val}(O^*)=\mathrm{ccon}(O^*)+\mathrm{cste}(O^*)$, escolhemos $\alpha\approx 1.35$ para minimizar o fator de aproximação, obtendo o seguinte resultado.

Teorema 8.11. *O Algoritmo 8.1 é um algoritmo $\beta=4.7$-aproximado para o Rent-or-Buy.*

8.5 Análise da marcação de independência limitada

Iremos apresentar alguns resultados úteis na Seção 8.5.1, que serão utilizados depois na Seção 8.5.2 para analisar a marcação com independência limitada comparada com a marcação totalmente independente.

8.5.1 Ferramentas de desaleatorização

Neste seção, apresentaremos alguns teoremas que serão úteis na análise da Seção 8.5.2. Muitos desses resultados serão somente enunciados, e utilizados como "caixa-preta", suas

demonstrações não fazem parte do escopo deste documento, e o leitor interessado deve recorrer às referências bibliográficas.

O seguinte teorema é um resultado que é utilizado para derivar uma desigualdade análoga à desigualdade de Chernoff, mas para marcações com independência limitada. Este resultado aparece no trabalho de Bellare e Rompel como Lema A.4 em [2].

Teorema 8.12 ([2])**.** *Seja $t \geq 2$ um inteiro par. Suponha que $X_1, \ldots, X_n$ são variáveis aleatórias t-a-t independentes, assumindo valores em $[0, 1]$. Seja $X = X_1 + \ldots + X_n$ e $\mu = E[X]$, e seja $A > 0$. Então*

$$E[(X - \mu)^t] \leq C_t \cdot (t\mu + t^2)^{t/2},$$

onde $C_t = 2\sqrt{\pi t} \cdot e^{1/6t} \cdot (5/(2e))^{t/2} \leq 8$.

Vamos introduzir algumas notações.

- Para cada grupo D_j, seja $Z_{j,1}, Z_{j,2}, \ldots, Z_{j,M}$ as variáveis aleatórias que indicam se uma demanda $i \in D_j$ foi marcada pelo algoritmo, considerando que essas demandas estão em *ordem não decrescente de distância de f_j*. Em outras palavras, $Z_{j,1} = 0$ se a demanda mais próxima a f_j não foi marcada, $Z_{j,1} = 1$ caso contrário. Analogamente $Z_{j,M} = 0$ se a demanda mais distante de f_j não foi marcada, $Z_{j,M} = 1$ caso contrário.
- Seja A algum conjunto de pares ordenados $\{(j, k)\}$. Então $N(A)$ é a variável aleatória que indica o evento "para todo $(j, k) \in A, Z_{j,k} = 0$". Além disso, $T(A)$ é a variável aleatória que indica $\sum_{(j,k)\in A} Z_{j,k}$. ("$N$" representa "nenhum", e "T" representa "total").

Seja

$$S^{(i)} = \sum_{A' \subseteq A : |A'| = i} \Pr\left[\bigcap_{(j,k)\in A'} Z_{j,k} = 1\right].$$

Vamos reescrever $\Pr[N(A) = 1]$ na sua expansão do *princípio da inclusão-exclusão* (ver definição no Apêndice A). Temos

$$\begin{aligned}
\Pr[N(A) = 1] &= \Pr\left[\bigcap_{(j,k)\in A} Z_{j,k} = 0\right] \\
&= 1 - \Pr\left[\bigcup_{(j,k)\in A} Z_{j,k} = 1\right] \\
&= 1 - S^{(1)} + S^{(2)} - \ldots + (-1)^i S^{(i)} \ldots (-1)^{|A|} S^{(|A|)}.
\end{aligned}$$

Iremos agora apresentar alguns limitantes superiores para $\mathrm{E}[N(A)]$.

Lema 8.13. *Seja t_1 um inteiro par positivo tal que $t_1 \leq t$ (lembrando que a marcação é feita de maneira t-a-t independente). Seja IE(t_1, A) a variável aleatória que representa a expansão do princípio da inclusão-exclusão da variável aleatória $N(A)$ truncada no t_1-ésimo nível, i.e.,*

$$IE(t_1, A) = 1 - S^{(1)} + S^{(2)} - \ldots + (-1)^{t_1} S^{(t_1)}$$

Então, para qualquer conjunto A de pares ordenados $\{(j,k)\}$, temos

(a) $N(A) \leq IE(t_1, A)$

(b) $E[IE(t_1, A)] \leq (1 - \alpha/M)^{|A|} + \left(\frac{e\alpha|A|}{Mt_1}\right)^{t_1}$.

Demonstração. Vamos provar a parte **(a)**. A variável $N(A)$ assume valores 0 e 1. Se $N(A) = 0$, claramente $N(A) \leq \text{IE}(t_1, A)$, pois a expansão do princípio da inclusão-exclusão nunca assume valor negativo em qualquer parte do truncamento. Caso contrário, se $N(A) = 1$, todos os $Z_{j,k} = 0, \forall (j,k) \in A$ e conseqüentemente todos termos $S^{(i)}$ da expansão são iguais a 0. Assim, $\text{IE}(t_1, A) = 1 = N(A)$.

A parte **(b)** segue do enunciado e da prova do Teorema 2 em [7]. ■

Lema 8.14. *Seja t_1 um inteiro par positivo tal que $t_1 \leq t$ (lembrando que a marcação é feita de maneira t-a-t independente). Seja NCM(t_1, A) o "momento central normalizado",i.e.,*

$$NCM(t_1, A) = \frac{(T(A) - \alpha|A|/M)^{t_1}}{(\alpha|A|/M)^{t_1}}.$$

Então, para qualquer conjunto A de pares ordenados $\{(j,k)\}$, temos

(a) $N(A) \leq NCM(t_1, A)$

(b) *se* $4 \leq t_1 \leq \alpha|A|/M$ *então* $E[NCM(t_1, A)] \leq 8 \cdot (2t_1)^{t_1/2} \cdot (\alpha|A|/M)^{-t_1/2}$.

Demonstração. Vamos provar a parte **(a)**. A variável $N(A)$ assume valores 0 e 1. Se $N(A) = 1$, então $T(A) = 0$, portanto é igual

$$\begin{aligned} \text{NCM}(t_1, A) &= \frac{(T(A) - \alpha|A|/M)^{t_1}}{(\alpha|A|/M)^{t_1}} \\ &= \frac{(-\alpha|A|/M)^{t_1}}{(\alpha|A|/M)^{t_1}}, \end{aligned}$$

e como t_1 é par, temos que $\text{NCM}(t_1, A) = 1$, portanto, $N(A) = 1 = \text{NCM}(t_1, A)$. Caso contrário, se $N(A) = 0, T(A) > 0$, e como t_1 é par, NCM assume um valor positivo, portanto $N(A) = 0 < \text{NCM}$.

Para a parte **(b)**, temos

$$\mathrm{E}[\mathrm{NCM}(t_1, A)] = \mathrm{E}\left[\frac{(T(A) - \alpha|A|/M)^{t_1}}{(\alpha|A|/M)^{t_1}}\right].$$

Como a marcação foi feita t-a-t independente, e estamos supondo $t_1 \leq t$, então podemos reescrever

$$\begin{aligned}\mathrm{E}[\mathrm{NCM}(t_1, A)] =& \frac{\mathrm{E}[(T(A) - \alpha|A|/M)^{t_1}]}{\mathrm{E}[(\alpha|A|/M)^{t_1}]} \\ =& \frac{\mathrm{E}[(T(A) - \mathrm{E}[T(A)])^{t_1}]}{(\alpha|A|/M)^{t_1}},\end{aligned}$$

onde a última igualdade vem do fato que a esperança de uma constante é a própria constante e $\mathrm{E}[T(A)] = \alpha|A|/M$. Usando o Teorema 8.12 e o fato que o termo $t\mu + t^2$ é no máximo $2t\mu$ se $t \leq \mu$, temos

$$\begin{aligned}\mathrm{E}[\mathrm{NCM}(t_1, A)] =& \frac{\mathrm{E}[(T(A) - \mathrm{E}[T(A)])^{t_1}]}{(\alpha|A|/M)^{t_1}} \\ \leq& \frac{8 \cdot (2t_1)^{t_1/2} \cdot \left(\frac{\alpha|A|}{M}\right)^{t_1/2}}{\left(\frac{\alpha|A|}{M}\right)^{t_1}} \\ =& \frac{8 \cdot (2t_1)^{t_1/2}}{\left(\frac{\alpha|A|}{M}\right)^{t_1/2}}\end{aligned}$$

e o lema segue. ■

8.5.2 Análise

Seja ϵ uma constante positiva qualquer em $(0, 1)$. Iremos provar que se as demandas são marcadas de maneira t-a-t independentes, onde $t = a\log(1/\epsilon)$ para uma constante a apropriadamente grande, então o fator de aproximação esperado é no máximo $1 + \epsilon$ vezes o fator obtido em nossa análise com marcação totalmente independente da Seção 8.4.1.

Pela parte **(a)** do Lema 8.9, podemos continuar usando o limitante superior encontrado neste lema para o custo da árvore de Steiner, mesmo sob a marcação t-a-t independente, o problema está com o limitante do custo de conexão. Iremos mostrar que o custo total de conexão esperado do experimento da Seção 8.4.1 muda muito pouco sob a marcação com independência limitada.

Relembrando o experimento da Seção 8.4.1. O custo total de conexão é uma variável aleatória que é a soma de três quantidades: (i) o valor determinístico $\sum_l C'_l$, que representa

o custo de todas demandas em D_l se conectarem a f_l, (ii) o valor que corresponde ao custo de ir até algum D_i, e (iii) o custo total pago por conectar f_i até a demanda marcada mais próxima em D_i, uma vez que D_i foi identificado como o grupo mais próximo. Iremos somente mostrar que o custo total esperado de (iii) é multiplicado por no máximo $(1+\epsilon)$ pela nossa marcação t-a-t independente, onde $t = b\log(1/\epsilon)$, os argumentos para o termo (ii) são análogos, e de fato, mais simples. Para ser mais específico, iremos mostrar o seguinte. Seja i o índice de algum D_i qualquer fixo. Iremos mostrar que o valor esperado da variável aleatória

$$\phi = \sum_{j=1}^{i} MX_jX_{j+1}\ldots X_{i-1}\cdot(1-X_i)A_i$$

é multiplicado por no máximo $(1+\epsilon)$[2].

Iremos agora mostrar como usar os Lemas 8.13 e 8.14 para limitar superiormente $\mathrm{E}[\phi]$. Sejam $a_1, a_2, \ldots, a_M$ as distâncias das demandas em D_i até f_i escritas em ordem *não-decrescente*. Então, expandindo a parte "$(1-X_i)A_i$" de ϕ, temos que

$$\phi = \sum_{j=1}^{i} MX_jX_{j+1}\ldots X_{i-1}\cdot\left[\sum_{u=1}^{M} a_uZ_{i,u}\cdot\prod_{l=1}^{u-1}(1-Z_{i,l})\right].$$

Fixe u qualquer, e seja

$$z_u = Z_{i,u}\cdot\prod_{l=1}^{u-1}(1-Z_{i,l})\cdot\sum_{j=1}^{i} MX_jX_{j+1}\ldots X_{i-1}.$$

Nossa intenção é mostrar que a esperança $\mathrm{E}[z_u]$ é multiplicada por no máximo $(1+\epsilon)$ em nossa marcação t-a-t independente, quando comparada com a marcação totalmente independente.

Para $j = 0, 1, \ldots, i-1$, defina $A_j = \{(r,s) : (i-j) \le r \le (i-1), 1 \le s \le M\}$. Assim, temos

$$z_u = Z_{i,u}\cdot N(\{(i,l) : 1 \le l \le u-1\})\cdot\sum_{j=0}^{i-1} N(A_j).$$

Sejam

- b_1 uma constante suficientemente grande,
- b_2 o menor inteiro *par* tal que $b_2 \ge 2e\log(1/\epsilon)$,
- t_1 o menor inteiro *par* tal que $t_1 \ge 2eb_1\log(1/\epsilon)$,

[2]Como i é fixo, não iremos subescrever ϕ como ϕ_i, esta observação também vale para algumas definições abaixo

- t_2 o maior inteiro *par* tal que $t_2 \leq b_1 \log(1/\epsilon)/4$.

Temos dois casos a considerar: (I) $b_1 \log(1/\epsilon) \leq i-1$ ou (II) $b_1 \log(1/\epsilon) > i-1$, iremos começar com o primeiro caso, que é o mais difícil.

Caso I: $b_1 \log(1/\epsilon) \leq i-1$. Neste caso, temos que dividir a expressão para z_u em duas somas, uma para os j's "pequenos", e outra para os j's "grandes":

$$\begin{aligned} z_u = & Z_{i,u} \cdot N(\{(i,l) : 1 \leq l \leq u-1\}) \cdot \sum_{j \leq b_1 \log(1/\epsilon)} N(A_j) \quad + \\ & Z_{i,u} \cdot N(\{(i,l) : 1 \leq l \leq u-1\}) \cdot \sum_{j > b_1 \log(1/\epsilon)} N(A_j). \end{aligned}$$

Agora, usando os Lemas 8.13 e 8.14, podemos sempre limitar a equação acima pela soma das seguintes duas variáveis aleatórias:

$$Z_{i,u} \cdot \mathrm{IE}(b_2, \{(i,l) : 1 \leq l \leq u-1\}) \cdot \sum_{j \leq b_1 \log(1/\epsilon)} \mathrm{IE}(t_1, A_j), \tag{8.9}$$

e

$$Z_{i,u} \cdot \mathrm{IE}(b_2, \{(i,l) : 1 \leq l \leq u-1\}) \cdot \sum_{j > b_1 \log(1/\epsilon)} \mathrm{NCM}(t_2, A_j), \tag{8.10}$$

Se nós expandirmos as esperanças dessas duas variáveis aleatórias usando linearidade da esperança, obteremos termos que são produtos de variáveis aleatórias Z, e além disso, o número de fatores em cada termo é no máximo $1+b_2+t_1$ no primeiro termo e $1+b_2+t_2$ no segundo termo. Portanto, se escolhermos t como sendo $t = 1+b_2+t_1$ (lembrando que $t_1 \leq t_2$, como definimos anteriormente), então a esperança dessas duas variáveis aleatórias fica

$$\mathrm{E}[Z_{i,u}] \cdot \mathrm{E}[\mathrm{IE}(b_2, \{(i,l) : 1 \leq l \leq u-1\})] \cdot \sum_{j \leq b_1 \log(1/\epsilon)} \mathrm{E}[\mathrm{IE}(t_1, A_j)], \tag{8.11}$$

e

$$\mathrm{E}[Z_{i,u}] \cdot \mathrm{E}[\mathrm{IE}(b_2, \{(i,l) : 1 \leq l \leq u-1\})] \cdot \sum_{j > b_1 \log(1/\epsilon)} \mathrm{E}[\mathrm{NCM}(t_2, A_j)], \tag{8.12}$$

respectivamente. Usando novamente os Lemas 8.13 e 8.14 para limitar esses valores, veremos que escolhendo a constante b_1 grande suficiente, a esperança $E[z_u]$ é no máximo $(1+\epsilon)$ vezes o que seria se utilizássemos a marcação totalmente independente.

A Expressão (8.11) é no máximo

$$\frac{\alpha}{M} \cdot \left((1-\alpha/M)^{u-1} + \left(\frac{e\alpha(u-1)}{Mb_2} \right)^{b_2} \right) \cdot \sum_{0 \leq j \leq b_1 \log(1/\epsilon)} \left((1-\alpha/M)^{Mj} + (e\alpha j/t_1)^{t_1} \right);$$

i.e., no máximo

$$\frac{\alpha}{M} \cdot \left((1-\alpha/M)^{u-1} + (e\alpha/b_2)^{b_2}\right) \cdot \sum_{0 \le j \le b_1 \log(1/\epsilon)} \left((1-\alpha/M)^{Mj} + (e\alpha b_1 \log(1/\epsilon)/t_1)^{t_1}\right).$$

Da mesma forma, a Expressão (8.12) é no máximo

$$\frac{\alpha}{M} \cdot \left((1-\alpha/M)^{u-1} + (e\alpha/b_2)^{b_2}\right) \cdot \sum_{j > b_1 \log(1/\epsilon)} \left(8 \cdot (2t_2/(\alpha j))^{t_2/2}\right).$$

Portanto, se as demandas foram marcados usando variáveis t-a-t independentes, então $\mathrm{E}[z_u]$ é no máximo $(\alpha/M) \cdot \left((1-\alpha/M)^{u-1} + (e\alpha/b_2)^{b_2}\right)$ vezes

$$\sum_{0 \le j \le b_1 \log(1/\epsilon)} \left((1-\alpha/M)^{Mj} + (e\alpha b_1 \log(1/\epsilon)/t_1)^{t_1}\right) + \sum_{j > b_1 \log(1/\epsilon)} \left(8 \cdot (2t_2/(\alpha j))^{t_2/2}\right).$$

Por outro lado, se usamos a marcação totalmente independente, temos

$$\mathrm{E}[z_u] = \frac{\alpha}{M} \cdot (1-\alpha/M)^{u-1} \cdot \sum_{j=0}^{i-1} (1-\alpha/M)^{Mj}. \tag{8.13}$$

Relembrando as definições de b_1, b_2, t_1 e t_2, é fácil de verificar que se b_1 é escolhido como uma constante suficientemente grande, então a primeira expressão é no máximo $(1+\epsilon)$ a segunda expressão. Basta atentar ao fato que, ao substituir os valores b_1, b_2, t_1 e t_2, alguns termos das equações acima ficam tão pequenos quanto queiramos, dependendo do valor ϵ escolhido. Isso completa a análise do Caso I.

Caso 2: $b_1 \log(1/\epsilon) > i - 1$. Seguimos a mesma análise do Caso I, mas não consideramos os casos dos j's "grandes". Assim, se usarmos a marcação com independência limitada, o limitante para $\mathrm{E}[z_u]$ é mais simples, é no máximo

$$\frac{\alpha}{M} \cdot \left(\left(1 - \frac{\alpha}{M}\right)^{u-1} + \left(\frac{e\alpha}{b_2}\right)^{b_2}\right) \cdot \sum_{0 \le j \le i-1} \left((1-\alpha/M)^{Mj} + (e\alpha b_1 \log(1/\epsilon)/t_1)^{t_1}\right).$$

Novamente, isso é no máximo $(1+\epsilon)$ o valor da Equação (8.13).

8.6 Extensões para outros casos

8.6.1 Caso com custos nas facilidades

Para o problema Connected Facility Location, podemos usar a seguinte variação do Algoritmo 8.1:

Algoritmo 8.2: CFL-Simples

Entrada: grafo $G = (V, E)$, demandas $D \subseteq V$, facilidades $F \subseteq V$, parâmetro $M > 1$;

Saída: árvore de Steiner T, facilidades abertas F'';

P1 Com probabilidade $1/M$, marque cada demanda $i \in F$. Seja $F' \subseteq F$ o conjunto das demandas marcadas.

P2 Construa uma solução do Facility Location ρ_{FL}-aproximada com custos de abertura de facilidade e as distâncias originais, mas com o valor das facilidades modificado: se $i \in F'$ então $f_i = M$, caso contrário, se $i \notin F'$ então $f_i = 0$. Seja F'' o conjunto de facilidades abertas retornadas por esse algoritmo.

P3 Construa uma árvore de Steiner ρ_{ST}-aproximada em $F'' \cup \{r\}$ e *compre* as arestas dessa árvore.

P4 Conecte cada demanda $j \in D$ à sua facilidade aberta mais próxima $i(j)$ em F''.

devolva T, F''

A análise de custo é bastante parecida com a que fizemos para o Rent-or-Buy, nós iremos somente esboçar as idéias principais. Sejam respectivamente $\text{cabr}(O^*), \text{ccon}(O^*)$ e $\text{cste}(O^*)$ o custos de abertura das facilidades, custo de conexão e o custo da árvore de Steiner de uma solução ótima O^*. Note que as facilidades da solução ótima formam uma solução viável para a instância do Facility Location no Passo C2, com custo esperado $(\text{cabr}(O^*) + \text{ccon}(O^*))$, e portanto pagamos

$$A_1 = \rho_{FL}(\text{cabr}(O^*) + \text{ccon}(O^*))$$

no valor esperado. Uma prova parecida com a do Lema 8.9 diz que a árvore de Steiner que conecta essas facilidades tem custo esperado $(\text{cste}(O^*) + \text{ccon}(O^*) + A_1)$, e portanto a árvore construída no Passo C3 custa no máximo ρ_{ST} vezes esse custo, ou seja

$$A_2 = \rho_{ST}(\text{cste}(O^*) + \text{ccon}(O^*) + \rho_{FL}(\text{ccon}(O^*) + \text{cabr}(O^*))).$$

Finalmente, temos que limitar os custos de conexão do Passo C4. Para isso, podemos fazer uma análise similar à análise feita na Seção 8.4.1, mas além do custo esperado $2\text{ccon}(O^*) + \frac{2e}{e-1}\text{cste}(O^*)$ que pagamos, temos também que pagar o custo adicional de ir das demandas marcadas em F' às facilidades em F'': isto tem custo esperado A_1. Portanto, o custo esperado no Passo C4 é

$$A_3 = 2\text{ccon}(O^*) + \frac{2e}{e-1}\text{cste}(O^*) + (\text{ccon}(O^*) + \text{cabr}(O^*))\rho_{FL}.$$

Somando as três expressões acima, limitamos o custo esperado como no máximo

$$\rho_{FL}(2+\rho_{ST})\text{cabr}(O^*) + (1+\rho_{FL})(2+\rho_{ST})\text{ccon}(O^*) + (\rho_{ST} + 2e/(e-1))\text{cste}(O^*).$$

Usando os melhores fatores de aproximação para $\rho_{FL} = 1.52$ [24] e $\rho_{ST} = 1.55$ [33], obtemos um fator de aproximação 8.94 para o Connected Facility Location. Uma aproximação melhor pode ser obtida se no Passo C1 escolhermos as demandas com probabilidade α/M, o que requer uma nova análise para determinar qual α minimiza a razão de aproximação.

Para obtermos custos-compartilhados cross-monotonic, podemos usar $\rho_{FL} = 3$, como descrito no Capítulo 6 e sua função de cost-sharing respectiva ξ_{FL}; e $\rho_{ST} = 2$, como vimos no Capítulo 5 e sua função de cost-sharing respectiva ξ_{ST}. Deste modo, obtemos um fator de aproximação $\beta = 16$, a o função de cost-sharing para cada demanda j será

$$\xi(D, j) = \frac{1}{\beta}\text{E}[M \cdot \xi_{ST}(F' \cup \{r\}, j) + M \cdot \xi_{FL}(F', j) + l(F' \cup \{r\}, j)],$$

que nos fornece um cost-sharing cross-monotonic β-aproximado para o Connected Facility Location.

8.6.2 Caso com demandas não uniformes

Nas seções anteriores, estávamos supondo que cada uma das demandas $j \in D$ tinha peso $d_j = 1$,i.e., a demanda só iria enviar à raiz uma unidade de peso. Além disso, assumimos que só existia um jogador em cada vértice em D. Vamos mostrar resumidamente como essas suposições podem ser descartadas. Poderíamos, ao descartar essas suposições, tentar obter melhorias nas constantes, a fim de otimizar os resultados, no entanto, vamos só esboçar as idéias principais para descartar essas suposições.

Seja v um vértice que tem n_v jogadores. Para descartar a suposição de um único jogador em cada vértice, criamos n_v novos vértices, ligamos cada um deles em v usando arestas de custo zero a colocamos cada jogador de v nesses novos vértices.

Para tratar demandas arbitrárias d_j para uma demanda j, vamos primeiramente olhar dois casos particulares separadamente, que garante que os valores de d_j estejam dentro de um fator multiplicativo de n^2 um do outro.

Vértice pesados: Se o peso $d_j \geq M$, podemos marcar *deterministicamente* o vértice j no Passo C1. É possível mostrar que uma solução ótima irá conectar esses vértices na árvore de Steiner, e portanto, fazer essa marcação não acarreta em perda na qualidade da solução.

Vértice leves: Se o peso $d_j \leq M/n^2$, então não marcamos j no Passo C1: note que o peso total $\sum_{j \in V} d_j$ de tais vértices é no máximo $n \cdot M/n^2 = M/n$. Assim, o custo

esperado da árvore de Steiner somente diminui quando não marcamos estes j's. Uma pequena variação na prova do Teorema 8.11 mostra que o custo esperado de atribuição cresce somente de um fator de $(1 + 1/n)$.

Após tratarmos destes dois casos, podemos aplicar uma escala e assumir que os pesos d_j estão entre $[n, n^3]$ e que o parâmetro M é maior que n^3. Assim, arredondamos para baixo os pesos d_j, fazendo $\lfloor d_j \rfloor$: isto diminui cada peso de um fator no máximo $(1 + 1/n)$, e portanto altera somente os termos de baixa ordem na garantia da aproximação. Finalmente, substituímos cada jogador j por $\lfloor d_j \rfloor$ jogadores. É possível modificar as provas da Seção 8.4 para mostrar que o algoritmo tem fator de aproximação constante mesmo com pesos arbitrários nas demandas, nos fornecendo também os cost-shares.

Capítulo 9

Conclusão e considerações finais

Apresentamos, nesta dissertação, resultados recentes sobre algoritmos de aproximação para problemas de projeto de redes e, além disso, introduzimos o assunto de compartilhamento de custos, aplicando as técnicas de algoritmos de aproximação para obter resultados de compartilhamento de custos. O assunto de compartilhamento de custos é um assunto muito recente, e já tornou-se alvo de uma parcela significativa da atual pesquisa na área de computação teórica.

Foram apresentadas duas principais técnicas durante este trabalho: primal-dual e probabilística. Muitas outras abordagens podem ser utilizadas para a obtenção de resultados novos. Como a idéia é uma "mistura" entre as áreas de algoritmos de aproximação e compartilhamento de custos, acreditamos que muitas outras técnicas, já utilizadas em algoritmos de aproximação, possam ser aplicadas em compartilhamento de custos. De fato, alguns resultados recentes apontam nesta direção.

Em particular, nossa intuição nos diz que as duas técnicas aqui apresentadas ainda têm muito a oferecer. No caso da técnica primal-dual que foi apresentada no Capítulo 6, esta pode ser aplicada de forma genérica. Isto significa que um grande número de problemas que já admitiam soluções aproximadas através desta técnica, podem agora ser aproveitados para obtenção de novos resultados. Além disso, há diversas variações nas técnicas de algoritmos de aproximação primais-duais e, desta forma, não estaremos somente limitados ao método específico aqui apresentado. Por exemplo, o método *dual-fitting* é uma variante da técnica primal-dual que aparenta ser bastante promissor.

Da mesma forma que a técnica primal-dual, a técnica probabilística nos parece bem atraente na busca por novos resultados. A simplicidade com que os algoritmos são desenvolvidos é um fator positivo desta técnica. Além disso, intuitivamente falando, a esperança parece ser uma boa aliada na obtenção de funções de compartilhamento de custo que têm a propriedade de ser cross-monotonic. Isto porque, ao adicionar mais usuários no sistema, a esperança é que o custo total diminua devido a um número maior de pessoas estar pre-

sente. Infelizmente nem sempre isto funciona, mas os resultados apresentados apontam evidências fortes para a utilização desta idéia.

Uma questão interessante diz respeito a obtenção de resultados negativos em compartilhamento de custos. Algumas de nossas tentativas na obtenção de resultados novos encontraram limitações decorrentes da impossibilidade de aproximação das funções de cost-sharing. Nestes se encaixam, por exemplo, a tentativa de obtenção de uma função de cost-sharing cross-monotonic para o problema do emparalhamento perfeito de peso mínimo utilizando a técnica primal-dual e a melhoria do fator de aproximação do problema Facility Location, apresentado no Capítulo 6. Um dos poucos trabalhos sobre resultados de inaproximabilidade é de Immorlica et.al. [15], que aponta limitações na aproximação de funções de cost-sharing para diversos problemas, incluindo os problemas Facility Location, cobertura por vértices, cobertura por conjuntos, etc.

Com o conhecimento adquirido nos estudos que resultaram nesta dissertação, esperamos que estes sejam úteis futuramente na obtenção de resultados novos. Além disso, esperamos que a dissertação seja um bom guia introdutório para os interessados no assunto de compartilhamento de custos e sua intersecção com a área de algoritmos aproximados.

Apêndice A

Espaços amostrais pequenos

Neste apêndice, traremos definições que serão úteis para o entendimento da construção de espaços amostrais pequenos. Além disso, mostramos como é feita a construção utilizada no Capítulo 8.

Definição A.1. Um *corpo* é uma estrutura algébrica que contém um conjunto de elementos satisfazendo um conjunto de operações e relações. Tais operações e relações satisfazem os axiomas descritos na tabela abaixo:

-	Adição	Multiplicação
Comutatividade	$a + b = b + a$	$ab = ba$
Associatividade	$(a + b) + c = a + (b + c)$	$(ab)c = a(bc)$
Distribuitividade	$a(b + c) = ab + ac$	$(a + b)c = ac + bc$
Identidade	$a + 0 = a = 0 + a$	$a \cdot 1 = a = 1 \cdot a$
Inversa	$a + (-a) = 0 = (-a) + a$	$aa^{-1} = 1 = a^{-1}a$ se $a \neq 0$

Por exemplo, os conjuntos do números reais formam um corpo, no entanto, o conjunto dos números inteiros não formam um corpo (pois não há inversa multiplicativa).

Definição A.2. Um *corpo finito* é um corpo com número de elementos finitos, também chamado de corpo de Galois. O tamanho de um corpo finito é sempre um número primo ou potência de um número primo. Para cada potência de primo, existe somente um (a menos de isomorfismo) corpo finito, denotado por $\mathrm{GF}(p^n)$. $\mathrm{GF}(p)$ é chamado de corpo primo de ordem p, e é o corpo das classes residuais módulo p, onde os p elementos são denotados $0, 1, \ldots, p - 1$. Qualquer operação válida neste corpo resulta em um elemento do próprio corpo.

Geralmente usa-se polinômios como sendo os elementos de um corpo finito, com os coeficientes desses polinômios sendo módulos de um primo. A representação por polinômios

é utilizada, ao invés de números, para evitar alguns problemas que fazem com que a representação dos elementos por números não atendam todas as propriedades axiomáticas de corpo.

Definição A.3. Dois eventos E e F são *independentes* se ambos tem probabilidade positiva e $\Pr[E|F] = \Pr[E]$ e $\Pr[F|E] = \Pr[F]$.

Uma moeda jogada duas vezes seguidas gera eventos independentes, pois o resultado do primeiro lançamento não influencia o resultado do segundo lançamento.

Teorema A.4. *Sejam E e F eventos, com $Pr[E], Pr[F] > 0$. Então E e F são independentes se e somente se $Pr[E \cap F] = Pr[E]Pr[F]$.*

Definição A.5. Um conjunto de n eventos $A = \{A_1, A_2, \ldots, A_n\}$ é dito *totalmente independente* se para qualquer subconjunto $A' = \{A_{i_1}, A_{i_2}, \ldots, A_{i_{|A'|}}\} \subseteq A$, onde $0 \leq |A'| \leq n$ temos que

$$\Pr[A_{i_1} \cap A_{i_2} \cap \ldots \cap A_{i_{|A'|}}] = \Pr[A_{i_1}]\Pr[A_{i_2}] \ldots \Pr[A_{i_{|A'|}}].$$

Definição A.6. Um conjunto de n eventos $A = \{A_1, A_2, \ldots, A_n\}$ é dito *t-a-t independente* se para qualquer subconjunto $A' = \{A_{i_1}, A_{i_2}, \ldots, A_{i_{|A'|}}\} \subseteq A$, onde $0 \leq |A'| \leq t$ temos que

$$\Pr[A_{i_1} \cap A_{i_2} \cap \ldots \cap A_{i_{|A'|}}] = \Pr[A_{i_1}]\Pr[A_{i_2}] \ldots \Pr[A_{i_{|A'|}}].$$

Na definição acima, se tivermos $|A'| > t$, nada é garantido com relação a independência dos eventos.

Um espaço amostral é dito *t-a-t independente* se qualquer combinação possível dos eventos são t-a-t independentes.

Proposição A.7 (Princípio da Inclusão-Exclusão)**.** *Sejam $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ eventos quaisquer. Então*

$$\begin{aligned}\Pr\left[\bigcup_{i=1}^{n} \varepsilon_i\right] = &\sum_i \Pr[\varepsilon_i] - \sum_{i<j} \Pr[\varepsilon_i \cap \varepsilon_j] \\ &+ \sum_{i<j<k} \Pr[\varepsilon_i \cap \varepsilon_j \cap \varepsilon_k] \\ &- \ldots + (-1)^{n+1} \sum_{i_1<i_2<\ldots<i_n} \Pr\left[\bigcap_{r=1}^{n} \varepsilon_{i_r}\right].\end{aligned}$$

A.1 Construção de espaços amostrais t-a-t independentes

Nesta seção vamos mostrar como construir o espaço amostral e as respectivas variáveis aleatórias utilizadas no Capítulo 8.

Vamos supor que temos um espaço amostral da forma $S^m = S \times S \times \ldots \times S$, onde S é um conjunto finito com distribuição uniforme. Sem perda de generalidade podemos pensar em S como sendo o conjunto dos inteiros $\{1, 2, \ldots, |S|\}$. Por exemplo, um dado com $|S|$ faces lançado m vezes gera um espaço amostral da forma S^m.

Qualquer elemento de S^m é uma sequência de m elementos de S, denotado por $(s_1, \ldots, s_m)$. Claramente, quaisquer dois componentes também têm distribuição uniforme. Note que, se o par $(x_1, x_2) \in X \times X$ é uniformemente distribuído em $X \times X$ então x_1 e x_2 são independentes e uniformemente distribuídos em X.

Vamos encontrar um subespaço Ω, menor que S^m, tal que a distribuição uniforme de Ω induz à distribuição uniforme em quaisquer par de pontos pertencentes a esse espaço, mas não induz distribuição uniforme a mais que dois pontos. As variáveis aleatórias nesse espaço são *par-a-par independentes*. Iremos depois extender para o caso t-a-t independentes. Na construção de Ω apresentada a seguir, iremos obter $|\Omega| = \max(|S|, m)^2$. Isto nos permite, de alguma maneira, "manter" a aleatoriedade.

A construção que é descrita em detalhes abaixo é um bom exemplo mas existem alguns detalhes técnicos que devem ser levados em consideração também. Esses detalhes técnicos serão citados, mas não serão explicados, para não prolongar demais o texto e desviar do objetivo deste apêndice.

Inicialmente, vamos supor que $m \leq |S|$. Esta restrição pode ser abandonada, e será comentada mais tarde. Iremos associar uma estrutura algébrica a S; por exemplo, tratando S como um corpo finito $\mathcal{F}$. Corpos finitos existem somente em tamanhos que são potências de números primos. O caso onde o tamanho de S não é uma potência de primo será comentado mais tarde.

Vamos então definir o espaço amostral Ω:

$$\begin{aligned}\Omega &\doteq \{\text{o conjunto de todos polinômios lineares em } \mathcal{F}\} = \{ax + b | (a, b) \in \mathcal{F}^2\} \\ &\equiv \{(a \cdot 1 + b, a \cdot 2 + b, \ldots, a \cdot m + b) | (a, b) \in \mathcal{F}^2\}.\end{aligned}$$

A primeira linha acima nos mostra uma representação sucinta de um conjunto de sequências de tamanho m em $\mathcal{F}$. A segunda linha nos mostra uma representação explícita das mesmas sequências.

Note que, dado um corpo $\mathcal{F}$, o espaço Ω pode ser facilmente amostrado: basta pegar aleatoriamente dois elementos do corpo e obtemos um elemento (sequência de tamanho m) de Ω.

Vamos denotar por $\Pr_{x \in_R X}$ a probabilidade sobre o conjunto X com distribuição uniforme nele. Além disso, seja $p_{a,b}(i) = a \cdot i + b$ (o polinômio linear em i com coeficientes a e b).

Proposição A.8. *Seja $[m]$ uma sequência de tamanho m. Para todo $i \neq j$ em $[m] \subseteq \mathcal{F}$ e para cada α, β em $\mathcal{F}$, temos*

$$\Pr_{(a,b) \in_R \mathcal{F}^2}\Big[p_{a,b}(i) = \alpha \cap p_{a,b}(j) = \beta\Big] = \frac{1}{|\mathcal{F}^2|}.$$

Em outras palavras, medir Ω uniformemente resulta em um espaço amostral com distribuição uniforme par-a-par independente.

Demonstração. Podemos notar que

$$\Pr_{(a,b) \in_R \mathcal{F}^2}\Big[p_{a,b}(i) = \alpha \cap p_{a,b}(j) = \beta\Big] = \frac{|\{(a,b) | p_{a,b}(i) = \alpha \cap p_{a,b}(j) = \beta\}|}{|\mathcal{F}^2|}.$$

Temos que mostrar que o numerador é igual a um. O numerador é igual ao número de pares (a, b) tal que o seguinte sistema de equações é satisfeito.

$$ai + b = \alpha$$
$$aj + b = \beta$$

Pensando em a e b como variáveis, e i, j, α e β como constantes, concluímos que se i não é igual a j, então esse sistema tem uma única solução pois a matriz

$$\begin{pmatrix} i & 1 \\ j & 1 \end{pmatrix}$$

tem posto cheio (todas as linhas são linearmente independentes). ■

Observe que a construção acima é independente do corpo que usamos, e podemos usar qualquer corpo que seja conveniente. Por exemplo, se o tamanho de S é primo, então a aritmética modular é uma boa escolha.

Note que a construção mostrada não é 3-a-3 independente. Não temos distribuição uniforme (e independente) em quaisquer *três* pontos, pois não há solução para o sistema de equações utilizado na prova da proposição.

De maneira mais geral, não é possível ter um espaço amostral t-a-t independente que é menor que $|S|^t$ pois quaisquer t pontos fixos são uniformemente distribuídos e todo valor de $(x_1, \ldots, x_t)$ tem probabilidade não nula de ocorrer, portanto é impossível ter menos que $|S|^t$ elementos nesse conjunto.

Usando a mesma idéia, podemos construir variáveis t-a-t independentes. Para isso, é suficiente considerar na construção de Ω os polinômios de grau menor que t. Assim temos

$$\begin{aligned}\Omega &\doteq \{\text{o conjunto de todos polinômios com grau menor que } t \text{ em } \mathcal{F}\}\\ &= \{\overline{c} = (c_0, \ldots, c_{t-1}) : p_{\overline{c}} = \sum_{i=0}^{t-1} c_i x^i\}\\ &\equiv \{(p_{\overline{c}}(1), \ldots, p_{\overline{c}}(m)) : \overline{c} \in \mathcal{F}^t\}.\end{aligned}$$

Novamente, a primeira linha acima nos mostra uma representação sucinta de um conjunto de sequências de tamanho m em $\mathcal{F}$. A última linha nos mostra uma representação explícita das mesmas sequências.

Lembrando que um espaço amostral é dito t-a-t independente se em quaisquer t pontos fixos nós temos distribuição uniforme. De maneira análoga à Proposição A.8, temos a seguinte proposição.

Proposição A.9. *Seja $[m]$ uma sequência de tamanho m. Para todo $i_1, \ldots, i_t$ distintos em $[m] \subset \mathcal{F}$ e para cada $\alpha_1, \ldots, \alpha_m$ em $\mathcal{F}$, temos*

$$Pr_{\overline{c} \in_R \mathcal{F}^t}\Big[\forall j = 1, \ldots, t, p_{\overline{c}}(i_j) = \alpha_{i_j}\Big] = \Big(\frac{1}{|\mathcal{F}|}\Big)^t.$$

Em outras palavras, medir Ω uniformemente resulta em um espaço amostral com distribuição uniforme t-a-t independente.

Demonstração. Da mesma forma que tínhamos na outra proposição,

$$\Pr_{\overline{c} \in_R \mathcal{F}^t}\Big[\forall j = 1, \ldots, t, p_{\overline{c}}(i_j) = \alpha_{i_j}\Big] = \frac{|\{\overline{c} | \forall j = 1, \ldots, t, p_{\overline{c}}(i_j) = \alpha_{i_j}\}|}{|\mathcal{F}|^t}$$

O numerador é igual ao número de soluços do seguinte sistema:

$$\begin{aligned} c_0 + c_1 i_1 + \ldots + c_{t-1} i_1^{t-1} &= \alpha_1\\ &\vdots\\ c_0 + c_1 i_t + \ldots + c_{t-1} i_t^{t-1} &= \alpha_t\end{aligned}$$

Novamente, devemos pensar nos c_i's como sendo as variáveis do seguinte sistema linear:

$$\begin{pmatrix} 1 & i_1 & i_1^2 & \ldots & i_1^{t-1}\\ 1 & i_2 & i_2^2 & \ldots & i_2^{t-1}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & i_t & i_t^2 & \ldots & i_t^{t-1}\end{pmatrix}\begin{pmatrix} c_0\\ c_1\\ \vdots\\ c_{t-1}\end{pmatrix} = \begin{pmatrix} \alpha_0\\ \alpha_1\\ \vdots\\ \alpha_{t-1}\end{pmatrix}$$

A matriz do sistema acima é uma matriz de Van der Monde, que tem a propriedade de ter posto cheio (pois $i_l \neq i_{l'}$ para todo $l \neq l'$). ■

Como afirmado no início desta seção, assumimos $m \leq |S|$. Além disso, o corpo só pode ter tamanho primo ou potência de primo. Esses detalhes não serão considerados no presente trabalho, mas são facilmente tratados, sendo que vários livros-textos comentam este assunto. Para o leitor interessado, recomendamos a leitura do capítulo sobre marcação limitada de [25], e do livro [30] que trata sobre corpos finitos.

Referências Bibliográficas

[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995. Versão preliminar no STOC'91.

[2] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 276–287, 1994.

[3] C. Bird. On cost allocation for a spanning tree: a game theoretic approach. *Networks*, 6:335–350, 1976.

[4] M. H. Carvalho, M. R. Cerioli, R. Dahab, P. Feofiloff, C. G. Fernandes, C. E. Ferreira, K. S. Guimarães, F. K. Miyazawa, J. C. Pina Jr., J. Soares, and Y. Wakabayashi. *Uma introdução sucinta a algoritmos de aproximação.* Editora do IMPA - Instituto de Matemática Pura e Aplicada, Rio de Janeiro–RJ, 2001.

[5] E. A. Choukhmane. Une heuristique pour le problème de l'arbre de steiner. *RAIRO Rech. Opér.*, (12):207–212, 1978.

[6] J. Edmonds. Optimum branchings. *J. Research of the National Bureau of Standards*, 71B:233–240, 1967.

[7] G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Velickovic. Approximations of general independent distributions. In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 10–16. ACM Press, 1992.

[8] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, 2001.

[9] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[10] J. Green, E. Kohlberg, and J. J. Laffont. Partial equilibrium approach to the free rider problem. *Journal of Public Economics*, 1976.

[11] A. Gupta, J. M. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a virtual private network: a network design problem for multicommodity flow. In *ACM Symposium on Theory of Computing*, pages 389–398, 2001.

[12] A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. *Proceedings of the thirty-fifth ACM symposium on Theory of computing*, 2003.

[13] A. Gupta, A. Srinivasan, and E. Tardos. Cost-sharing mechanisms for network design. In *APPROX-RANDOM*, pages 139–150, 2004.

[14] D. S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.

[15] N. Immorlica, M. Mahdian, and V. S. Mirrokni. Limitations of cross-monotonic cost sharing schemes. In *SODA*, pages 602–611, 2005.

[16] A. Iwainsky, E. Canuto, O. Taraszow, and A. Villa. Network decomposition for the optimization of connection structures. *Networks*, (16):205–235, 1986.

[17] K. Jain and V. V. Vazirani. Applications of approximation algorithms to cooperative games. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 364–372, 2001.

[18] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[19] S. Khuller and An Zhu. The general steiner tree-star problem. *Information Processing Letters*, 84:215–220, 2002.

[20] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, (15):141–145, 1981.

[21] C. Krick, H. Racke, and M. Westermann. Approximation algorithms for data management in networks. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 237–246, 2001.

[22] S. Leonardi and G. Schaefer. Cross-monotonic cost-sharing methods for connected facility location games. In *EC '04: Proceedings of the 5th ACM conference on Electronic commerce*, pages 242–243, 2004.

[23] R. Graham M. Garey and D. Johnson. The complexity of computing steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32:835–859, 1977.

[24] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *APPROX '02: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 229–242. Springer-Verlag, 2002.

[25] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[26] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[27] H. Moulin. Incremental cost-sharing: characterization by coalition strategy-proofness. *Social Choice and Welfare*, 16:279–320, 1999.

[28] H. Moulin and S. Shenker. Strategyproof sharing of submodular costs: Budget balance versus eficiency, 1997.

[29] J. Plesník. A bound for the steiner tree problem in graphs. *Math. Slovaca*, 1981.

[30] O. Pretzel. *Error-correcting codes and finite fields (student ed.)*. Oxford University Press, Inc., 1996.

[31] M. Pál and E. Tardos. Group strategyproof mechanisms via primal-dual algorithms. In *FOCS*, pages 584–593, 2003.

[32] K. Roberts. *Aggregation and Revelation of Preferences*, chapter The Characterization of Implementable Choice Rules, pages 321–348. North-Holland, Amsterdam, 1979.

[33] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 770–779, 2000.

[34] C. Swamy and A. Kumar. Primal-dual algorithms for the connected facility location problem. In *5th APPROX, LNCS*, pages 256–269, 2002.

[35] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2000.

[36] D. P. Williamson. Lecture notes on approximation algorithms. Technical Report RC 21409, T.J. Watson Research Center (IBM Research Division), Yorktown Heights, New York, 1998.

[37] D. P. Williamson. The primal-dual method for approximation algorithms. *Mathematical Programming Series B*, 91(3):447–478, 2002.

Índice Remissivo