Uma Introdução Sucinta a Algoritmos de Aproximação

Marcelo Henriques de Carvalho
Universidade Federal do Mato Grosso do Sul

Márcia Rosana Cerioli
Universidade Federal do Rio de Janeiro

Ricardo Dahab
Universidade Estadual de Campinas

Paulo Feofiloff
Cristina Gomes Fernandes
Carlos Eduardo Ferreira
Universidade de São Paulo

Katia Silva Guimarães
Universidade Federal de Pernambuco

Flávio Keidi Miyazawa
Universidade Estadual de Campinas

José Coelho de Pina Jr.
José Augusto R. Soares
Yoshiko Wakabayashi
Universidade de São Paulo

maio de 2001

Prefácio

O desenvolvimento de algoritmos de aproximação e de provas de inaproximabilidade é uma das linhas de pesquisa que mais têm crescido na área de otimização combinatória e teoria da computação. O objetivo deste texto é apresentar sucintamente uma visão sistemática das técnicas utilizadas no projeto de algoritmos de aproximação e dar uma idéia dos limites intrínsecos da aproximabilidade.

Ainda que os tópicos cobertos neste texto possam ser encontrados em outros livros [Hoc97, MPS98, ACG$^+$99, Vaz01], o presente texto se justifica pela escassez de material em língua portuguesa. Além disso, a seleção de tópicos e a profundidade com que certos assuntos foram tratados aqui são, para nosso conhecimento, inéditas.

Este texto é sucinto no sentido de que se limita a apresentar os algoritmos e suas análises, sem se alongar com comentários e exemplos. Mas cada capítulo contém notas bibliográficas e uma variedade de exercícios que complementam o texto. Esperamos que apesar de sucinto, o livro tenha um caráter didático e seja útil tanto aos que procuram uma introdução à área quanto aos que queiram estudar esse fascinante tema mais profundamente.

O assunto tem muitos pré-requisitos (teoria dos grafos, programação linear, teoria das probabilidades, complexidade computacional), que procuramos cobrir compactamente em apêndices. Recomendamos que o leitor recorra aos apêndices somente na medida do necessário, com a ajuda do índice.

Os capítulos 1 e 2 apresentam as definições básicas, um histórico da área e os primeiros algoritmos de aproximação para alguns problemas clássicos de otimização combinatória. Isoladamente, esses dois capítulos formam uma introdução ao assunto.

Os capítulos de 3 a 5 apresentam os métodos baseados em programação linear. A seção 3.1 e o capítulo 4 tratam de algoritmos simples, cuja análise depende apenas de conceitos e resultados básicos de programação linear. Ao leitor que tenha conhecimento superficial no assunto, recomendamos a leitura do apêndice C. A seção 3.2 descreve um algoritmo não-trivial cuja análise é longa e elaborada, ainda que não envolva conceitos avançados. O capítulo 5 apresenta uma técnica elegante e sofisticada, derivada do método primal-dual de programação linear, que se apóia no conceito de folgas complementares.

O capítulo 6 discute algoritmos probabilísticos por meio de um exemplo simples mas instrutivo. Um dos algoritmos apresentados neste capítulo utiliza programação linear. Acreditamos que o conteúdo do capítulo seja acessível a qualquer leitor que conheça os conceitos de variável aleatória e esperança, e tenha noções básicas de programação linear.

No capítulo 7, sobre programação semidefinida, descrevemos uma técnica sofisticada e relativamente recente que tem sido aplicada com sucesso a diversos problemas. Este capítulo usa todas as ferramentas apresentadas nos capítulos anteriores: generaliza a técnica de relaxações lineares e usa um tipo de arredondamento probabilístico bem mais sofisticado que os vistos no capítulo 6. O capítulo requer maturidade e familiaridade com os vários conceitos envolvidos.

O capítulo 8 descreve a classificação geral de problemas de otimização do ponto de vista da aproximabilidade. Também discute a posição que os diversos problemas tratados no texto ocupam nesta classificação. Este capítulo pode ser lido à parte pelos interessados em teoria da computação.

Este livro pode ser adotado em disciplinas de final da graduação ou de pós-graduação. Alguns dos problemas no capítulo 2, e talvez uma ou duas demonstrações de inaproximabilidade do capítulo 8, podem ser usados como um tópico em uma disciplina de graduação de análise de algoritmos. As definições básicas e dois ou três exemplos devem ser suficientes para dar uma idéia dos objetivos da área. Uma disciplina dedicada exclusivamente a algoritmos de aproximação tem como pré-requisitos noções de teoria dos grafos, programação linear e teoria de complexidade de algoritmos. O conteúdo completo do livro é adequado para uma disciplina de pós-graduação com duração de um semestre. Na graduação, pode-se optar, dependendo da maturidade e interesse dos alunos, por não cobrir o material da seção 3.2 e do capítulo 7, e cobrir apenas parcialmente o capítulo 8. Uma versão preliminar do livro foi utilizada, durante o primeiro semestre de 2001, em uma disciplina de graduação na UFRJ e em disciplinas de pós-graduação na USP e na UNICAMP.

O livro tem 11 autores, provenientes de cinco universidades espalhadas por três regiões do país: UFMS (1 autor), UFPE (1 autora), UFRJ (1 autora), UNICAMP (2 autores) e USP (6 autores). Todos participam do Projeto ProNEx *Complexidade de Estruturas Discretas* (`http://www.ime.usp.br/~yoshi/pronex/`), coordenado por Yoshiharu Kohayakawa, e a elaboração do livro fez parte das atividades do Projeto. Quatro dos autores atuaram como editores, cuidando da coordenação e dos detalhes de edição. Com tantos envolvidos, é quase impossível chegar a um consenso sobre qualquer questão específica. Pode-se dizer que cada parágrafo do texto desagrada algum dos autores; esperamos, ao menos, que não desagrade a todos os leitores! Os autores reuniram-se algumas vezes para discutir o texto, mas boa parte da interação e da redação aconteceu via internet. Foi um período de trabalho coletivo intenso.

Manteremos em `http://www.ime.usp.br/dcc/livros/aprox/` uma errata do livro e um endereço eletrônico para onde podem ser enviadas as comunicações de erros, dúvidas e sugestões.

Este não é o primeiro texto em língua portuguesa sobre o assunto: Katia S. Guimarães, uma das autoras desse livro, apresentou um curso sobre o assunto [Gui98] na *XVII Jornada de Atualização em Informática*, em 1997. Aquele material serviu como versão inicial para duas seções do presente livro.

Agradecemos a Celina M. H. Figueiredo (UFRJ), Sulamita Klein (UFRJ) e Luerbio Faria (UERJ) pela sua contribuição com uma primeira versão da seção 2.3. Agradecemos aos alunos que leram e criticaram versões preliminares de alguns capítulos. Somos gratos também ao Projeto ProNEx *Complexidade de Estruturas Discretas* pelo suporte financeiro para os encontros que se fizeram necessários. Em especial, agradecemos ao coordenador do projeto pelo incentivo e pela disposição em responder questões técnicas sobre algoritmos probabilísticos. Agradecemos também à Comissão Organizadora do 23º Colóquio Brasileiro de Matemática pela oportunidade.

Esperamos que os leitores se divirtam e aprendam tanto quanto os autores enquanto escreviam o livro!

Rio de Janeiro, São Paulo e Campinas, maio de 2001

M.R.C., P.F., C.G.F e F.K.M.
editores

Sumário

Capítulo 1

Introdução

Problemas de otimização têm o objetivo de encontrar um ponto ótimo (mínimo ou máximo) de uma função definida sobre um certo domínio. Os problemas de otimização combinatória têm domínio finito. Embora os elementos do domínio possam, em geral, ser facilmente enumerados, a idéia ingênua de testar todos os elementos na busca pelo melhor mostra-se inviável na prática pois o domínio é tipicamente muito grande.

Como exemplos clássicos de problemas de otimização combinatória podemos citar o problema do caixeiro viajante, o problema da mochila, o problema da cobertura mínima por conjuntos, o problema da floresta de Steiner e o problema da satisfatibilidade máxima. Todos surgem naturalmente em aplicações práticas, tais como o projeto de redes de telecomunicação e de circuitos VLSI, o empacotamento de objetos em *containers*, a localização de centros distribuidores, o escalonamento e roteamento de veículos, etc. Outras áreas de aplicação incluem a estatística (análise de dados), a economia (matrizes de entrada/saída), a física (estados de energia mínima), a biologia molecular (alinhamento de DNA e proteínas, inferência de padrões), etc.

O desenvolvimento de algoritmos de aproximação surgiu em resposta à dificuldade computacional de muitos dos problemas de otimização combinatória: em termos técnicos, muitos são NP-difíceis. Nessa situação, é razoável sacrificar a otimalidade em troca de uma aproximação de boa qualidade que possa ser eficientemente calculada. Esse compromisso entre perda de otimalidade e ganho em eficiência é o paradigma dos algoritmos de aproximação. Convém observar que um algoritmo de aproximação não é simplesmente uma heurística: ele garante encontrar, eficientemente, um elemento do domínio cujo valor guarda uma relação pré-estabelecida com o valor ótimo.

No início da década de 70, Garey, Graham e Ullman [GGU72], bem como Johnson [Joh74], formalizaram o conceito de algoritmo de aproximação. O conceito já estava implícito em um trabalho de Graham [Gra66] sobre um problema de escalonamento em máquinas paralelas e em um trabalho de Erdős [Erd67] sobre grafos bipartidos. Na década de 90, o estudo de algoritmos de aproximação passou a receber um tratamento mais sistemático, com a formalização e o uso de técnicas e ferramentas aplicáveis a toda uma gama de problemas.

É importante mencionar também o aparecimento de certos *resultados negativos* de aproximabilidade: para alguns problemas, aproximar é tão difícil quanto resolver. Em termos mais técnicos, alguns problemas não admitem algoritmos de aproximação com razão melhor que um certo limiar, a menos que P = NP. As teorias nessa direção foram impulsionadas na década de 90 pelas descobertas de Arora *et al.* [ALM+92, Aro95], que provaram resultados desse tipo para vários problemas usando caracterizações probabilísticas da classe NP.

O desenvolvimento de algoritmos de aproximação e de provas de inaproximabilidade é uma das linhas de pesquisa que mais cresceu ultimamente na área de otimização combinatória e teoria da computação. Esta observação encontra respaldo na grande quantidade de artigos de pesquisa que surgiram nos últimos anos (veja nossa lista de referências bibliográficas). Vários livros sobre o assunto também foram publicados recentemente: Ausiello *et al.* [ACG+99], Hochbaum [Hoc97], Mayr *et al.* [MPS98] e Vazirani [Vaz01]. Outro indício da efervescência da área é a grande quantidade de teses de doutorado concluídas na década de 90, algumas introduzindo teorias revolucionárias [Aro94, Blu91, Kan92, Tre96, Wil93].

1.1 Problemas de otimização

Um problema de otimização tem três ingredientes principais: um
$\mathrm{Sol}(I)$ conjunto de **instâncias**[1] (*instances*), um conjunto $\mathrm{Sol}(I)$ de **soluções viáveis** (*feasible solutions*) para cada instância I, e uma função que
$\mathrm{val}(S)$ atribui um número $\mathrm{val}(S)$ a cada solução viável S. O número $\mathrm{val}(S)$ é o **valor** de S. Quando o conjunto $\mathrm{Sol}(I)$ das soluções viáveis associado a uma instância I é vazio, dizemos que a instância é **inviável**; caso contrário, a instância é **viável**.

[1] É uma pena que o uso tenha imposto a tradução incorreta *instância* para o termo inglês *instance*.

Um problema **de minimização** está interessado nas soluções viáveis de valor mínimo, enquanto um problema **de maximização** está interessado nas soluções viáveis de valor máximo.[2] Quando uma dessas alternativas — mínimo ou máximo — está subentendida, dizemos simplesmente **valor ótimo** e problema **de otimização**. Uma solução viável cujo valor é ótimo é chamada **solução ótima** (*optimal solution*). O valor de qualquer das soluções ótimas de uma instância I será denotado por $\mathrm{opt}(I)$. Portanto, $\mathrm{opt}(I)$

$$\mathrm{opt}(I) := \mathrm{val}(S^*)\,,$$

onde S^* é uma solução ótima de I. É claro que esse número só está definido se a instância I é viável.

A título de exemplo, eis um problema de otimização bem conhecido: encontrar um circuito hamiltoniano de custo mínimo em um grafo com custos nas arestas. Uma instância desse problema consiste em um grafo G e uma função c que associa um número não-negativo a cada aresta de G. O conjunto das soluções viáveis de uma instância (G, c) é o conjunto de todos os circuitos hamiltonianos de G. O valor de um circuito hamiltoniano C é $\mathrm{val}(C) := \sum_{e \in C} c_e$.

1.2 Algoritmos de aproximação

Considere um problema de otimização em que $\mathrm{val}(S) \geq 0$ para toda solução viável S de qualquer instância do problema. Seja A um algoritmo que, para toda instância viável I do problema, devolve uma solução viável $A(I)$ de I. Se o problema é de minimização e

$$\mathrm{val}(A(I)) \;\leq\; \alpha\,\mathrm{opt}(I) \tag{1.1}$$

para toda instância I, dizemos que A é uma **α-aproximação** para o problema. O fator α é um número que pode depender de I. Dizemos que α é uma **razão de aproximação** (*approximation factor*) do algoritmo. É claro que $\alpha \geq 1$, uma vez que o problema é de minimização. No caso de problema de maximização, basta refazer a definição com

$$\mathrm{val}(A(I)) \;\geq\; \alpha\,\mathrm{opt}(I)$$

[2] A expressão "valor da solução" pode ser trocada por "custo da solução" no caso de problemas de minimização e por "peso da solução" no caso de problemas de maximização.

no lugar de (1.1). É claro que nesse caso $0 < \alpha \leq 1$. Um **algoritmo de aproximação** (*approximation algorithm*) é uma α-aproximação para algum α. Uma 1-aproximação para um problema de otimização é um **algoritmo exato** para o problema.

Observe que um algoritmo A é uma α-aproximação para um problema de minimização (maximização) se α é uma delimitação superior (inferior) para a razão entre $\mathrm{val}(A(I))$ e $\mathrm{opt}(I)$ para uma instância arbitrária I do problema. Como o valor de $\mathrm{opt}(I)$ é em geral tão difícil de calcular quanto uma solução ótima do problema, para demonstrar que um algoritmo é uma α-aproximação é essencial, como veremos, ter boas delimitações para o valor de $\mathrm{opt}(I)$.

A cada instância I de um dado problema está associado um número natural $\langle I \rangle$ que chamamos **tamanho** da instância. (Podemos imaginar que as instâncias, bem como as soluções viáveis, são cadeias de caracteres; nesse caso, $\langle I \rangle$ é o comprimento da cadeia de caracteres I.) Um algoritmo A para o problema é **polinomial** se existe um polinômio p tal que o consumo de tempo do algoritmo é limitado por $p(\langle I \rangle)$ para cada instância I. O conceito de algoritmo polinomial deve ser entendido como uma formalização da idéia de algoritmo eficiente. Se um problema é NP-difícil então é improvável que exista um algoritmo polinomial exato para o problema.

1.3 Notação básica

Aqui estabelecemos algumas convenções básicas de notação. Dada uma função c que associa um número a cada elemento e de um conjunto finito E, denotamos o valor de c em e por c_e. Para qualquer subconjunto
$c(\,)$ F de E, denotamos por $c(F)$ a soma dos valores de c nos elementos de F:

$$c(F) \;:=\; \textstyle\sum_{f \in F} c_f\,.$$

Essa convenção vale, em particular, quando c é um vetor indexado por E.

Os conjuntos dos números inteiros, racionais e reais são denotados
$\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ por $\mathbb{Z}$, $\mathbb{Q}$ e $\mathbb{R}$, respectivamente. Os subconjuntos desses conjuntos que contêm somente os números não-negativos são indicados por um "$_\geq$"; os que contêm somente os positivos são indicados por um "$_>$". Assim, por
$\mathbb{Q}_\geq$ exemplo, o conjunto dos racionais não-negativos é denotado por $\mathbb{Q}_\geq$ e o
$\mathbb{Q}_>$ dos racionais positivos por $\mathbb{Q}_>$.

Para outras convenções de notação, envolvendo grafos, vetores e matrizes, programação linear, probabilidades e complexidade computacional, consulte, se necessário, os apêndices A, B, C, D e E respectivamente.

CAPÍTULO 2

Algoritmos Clássicos

Neste capítulo descrevemos algoritmos de aproximação para quatro dos mais célebres problemas de otimização combinatória. Estes estão entre os primeiros algoritmos de aproximação mencionados na literatura. Cada um deles foi desenvolvido de forma independente, em função das características estruturais do problema em questão.

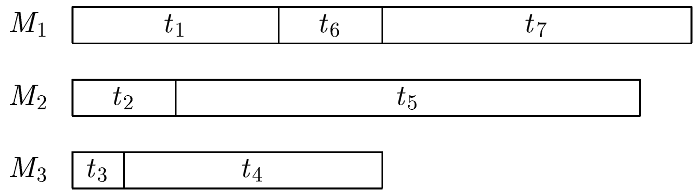
2.1 Escalonamento

Um problema bastante conhecido nos contextos de produção industrial e de sistemas operacionais é o de escalonamento (*scheduling*) de tarefas em máquinas. Estamos interessados em uma das versões mais simples do problema: dadas m máquinas idênticas e n tarefas com tempos de execução pré-determinados, encontrar uma atribuição das tarefas às máquinas que minimize o tempo máximo de operação de qualquer uma das máquinas (*makespan*).

Formalmente, o **problema do escalonamento em máquinas idênticas** (*multiprocessor scheduling problem*) consiste no seguinte:

> **Problema** ESCALONAMENTO (m, n, t): *Dados inteiros positivos m, n e um tempo t_i em $\mathbb{Q}_{\geq}$ para cada i em $\{1, \ldots, n\}$, encontrar uma partição $\{M_1, \ldots, M_m\}$ de $\{1, \ldots, n\}$ que minimize* $\max_j t(M_j)$.

De acordo com nossa convenção de notação, $t(M_j) := \sum_{i \in M_j} t_i$. Dizemos que uma partição de $\{1, \ldots, n\}$ em m blocos é um **escalonamento** e que o número $\max_j t(M_j)$ é o **custo** do escalonamento. Com essa terminologia, o problema pode ser formulado como: dados m, n e t,



t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6

Figura 2.1: Exemplo de escalonamento com 7 tarefas em 3 máquinas. A duração da tarefa i é t_i. O critério de Graham produz o escalonamento $\{\{1,6,7\},\{2,5\},\{3,4\}\}$, representado na figura. Esse escalonamento tem custo $t_1+t_6+t_7=12$. Um outro escalonamento é $\{\{1,7\},\{2,4,6\},\{3,5\}\}$, que tem custo $t_3+t_5=10$.

encontrar um escalonamento de custo mínimo.

Este problema é NP-difícil mesmo para duas máquinas, ou seja, quando $m=2$ [GJ79]. O primeiro algoritmo de aproximação para o problema foi descrito e analisado por Graham [Gra66] e usa um critério muito simples: alocar as tarefas uma a uma, destinando cada tarefa à máquina menos ocupada. Por esse critério, a escolha da máquina que vai receber determinada tarefa não depende dos tempos das tarefas que ainda não foram atribuídas a nenhuma máquina. Veja um exemplo na figura 2.1.

Algoritmo ESCALONAMENTO-GRAHAM (m, n, t)

para j de 1 a m faça $M_j \leftarrow \emptyset$
para i de 1 a n faça
    seja k uma máquina tal que $t(M_k)$ é mínimo
    $M_k \leftarrow M_k \cup \{i\}$
devolva $\{M_1, \ldots, M_m\}$

Claramente, ao final, $\{M_1, \ldots, M_m\}$ é uma partição de $\{1, \ldots, n\}$, ou seja, um escalonamento. Há duas delimitações simples para o custo $\mathrm{opt}(m,n,t)$ de um escalonamento ótimo: o tempo da tarefa mais longa e o custo que obteríamos se pudéssemos distribuir as tarefas por igual entre as m máquinas. Em fórmulas,

$$\mathrm{opt}(m,n,t) \geq \max_i t_i \qquad \text{e} \qquad \mathrm{opt}(m,n,t) \geq \tfrac{1}{m}\textstyle\sum_{i=1}^{n} t_i \,. \tag{2.1}$$

No exemplo da figura 2.1, o segundo escalonamento apresentado é ótimo, já que $\frac{1}{m}\sum_i t_i = 9{,}666\ldots$ Essas delimitações são usadas na análise do algoritmo.

O caso em que $n < m$ pode ser resolvido de maneira simples, atribuindo-se uma tarefa a cada máquina, assim no seguinte teorema nos restringimos ao caso em que $n \geq m$.

Teorema 2.1: *O algoritmo* ESCALONAMENTO-GRAHAM *é uma 2-aproximação polinomial para o* ESCALONAMENTO (m,n,t), *quando* $n \geq m$.

Demonstração: Seja τ o valor de $t(M_k)$ imediatamente antes da execução da linha 4 do algoritmo em uma iteração qualquer. É claro que $\tau \leq t(M_j)$ para todo j e, portanto, em virtude de (2.1),

$$\tau \;\leq\; \tfrac{1}{m}\textstyle\sum_{j=1}^{m} t(M_j) \;\leq\; \tfrac{1}{m}\textstyle\sum_{i=1}^{n} t_i \;\leq\; \mathrm{opt}(m,n,t)\,.$$

Assim, imediatamente depois da execução da linha 4 do algoritmo, temos $t(M_k) = \tau + t_i \leq 2\,\mathrm{opt}(m,n,t)$, pois $t_i \leq \mathrm{opt}(m,n,t)$ de acordo com (2.1). Essa delimitação vale no fim de cada iteração e portanto aplica-se a cada uma das máquinas. Logo, no fim da última iteração,

$$\max_j t(M_j) \leq 2\,\mathrm{opt}(m,n,t)\,.$$

Quanto ao tempo de execução, é fácil ver que o algoritmo consome tempo polinomial em n, já que $m \leq n$. Assim, o algoritmo é polinomial. □

Conforme já observamos, o algoritmo processa os dados sem conhecimento prévio da sua totalidade. Várias situações reais demandam esse tipo de abordagem, entre elas o escalonamento de tarefas em processadores e a alocação de páginas na memória primária de computadores. Algoritmos para tais aplicações devem ser rápidos e fornecer soluções viáveis cujo valor seja próximo do valor ótimo. O algoritmo ESCALONAMENTO-GRAHAM possui estas características e é conceitualmente muito simples. Esse foi o primeiro algoritmo de aproximação de que se tem notícia, o que o torna um marco nessa teoria.

2.2 Cobertura por conjuntos

O segundo problema que vamos discutir é uma abstração de vários problemas que são abordados ao longo deste texto.

Dada uma coleção finita $\mathcal{S}$ de conjuntos finitos, dizemos que uma subcoleção $\mathcal{T}$ de $\mathcal{S}$ **cobre** um conjunto finito E se todo elemento de E pertence a algum conjunto de $\mathcal{T}$. Neste caso, dizemos também que $\mathcal{T}$ é uma **cobertura** de E. O **problema da cobertura mínima por conjuntos** (*minimum set cover problem*) consiste no seguinte:

> **Problema** MINCC $(E, \mathcal{S}, c)$: *Dados um conjunto finito E, uma coleção finita $\mathcal{S}$ de conjuntos finitos que cobre E e um custo c_S em $\mathbb{Q}_{\geq}$ para cada S em $\mathcal{S}$, encontrar uma cobertura $\mathcal{T}$ de E que minimize $c(\mathcal{T})$.*

A exigência de que $\mathcal{S}$ cubra E é inócua: ela apenas exclui as instâncias inviáveis do problema. Chamamos o número $c(\mathcal{T})$ de **custo** da cobertura $\mathcal{T}$. Assim, o problema consiste em encontrar uma cobertura de custo mínimo.

O MINCC é NP-difícil mesmo quando cada conjunto em $\mathcal{S}$ não tem mais que três elementos [GJ79]. Uma estratégia gulosa simples para o problema consiste em selecionar repetidamente o conjunto em $\mathcal{S}$ que é mais "promissor" em termos de custo com relação ao número de elementos ainda não cobertos que ele contém. Essa estratégia dá origem ao seguinte algoritmo, proposto por Chvátal [Chv79], que apresentamos em sua versão recursiva.

Algoritmo MINCC-CHVÁTAL $(E, \mathcal{S}, c)$

se $E = \emptyset$
    então devolva $\emptyset$
    senão seja Z em $\mathcal{S}$ tal que $c_Z/|Z \cap E|$ é mínimo
        $E' \leftarrow E \setminus Z$
        $\mathcal{S}' \leftarrow \{ S \in \mathcal{S} : S \cap E' \neq \emptyset \}$
        seja c' a restrição de c a $\mathcal{S}'$
        $\mathcal{T}' \leftarrow$ MINCC-CHVÁTAL $(E', \mathcal{S}', c')$
        devolva $\{Z\} \cup \mathcal{T}'$

Claramente o algoritmo devolve uma cobertura de E. Há uma boa delimitação inferior para o valor ótimo do problema MINCC $(E, \mathcal{S}, c)$

relacionada ao algoritmo acima. Para mostrá-la, suponha que Z é um elemento de $\mathcal{S}$ tal que $c_Z/|Z \cap E| \leq c_S/|S \cap E|$ para todo S em $\mathcal{S}$. Então, para qualquer cobertura $\mathcal{T}$ de E,

$$\begin{aligned}
\frac{c_Z}{|Z \cap E|}|E| &\leq \frac{c_Z}{|Z \cap E|} \textstyle\sum_{S \in \mathcal{T}} |S \cap E| \\
&\leq \textstyle\sum_{S \in \mathcal{T}} \frac{c_S}{|S \cap E|}|S \cap E| \\
&= \textstyle\sum_{S \in \mathcal{T}} c_S \\
&= c(\mathcal{T})\,.
\end{aligned}$$

Segue daí que

$$\operatorname{opt}(E, \mathcal{S}, c) \geq |E|\, \frac{c_Z}{|Z \cap E|}\,. \tag{2.2}$$

A seguir, mostramos que o algoritmo MinCC-Chvátal é uma H_n-aproximação, onde $n := |E|$ e H_n é o número harmônico: H_n

$$H_n := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\,.$$

Teorema 2.2: *O algoritmo* MinCC-Chvátal *é uma H_n-aproximação polinomial para o* MinCC $(E, \mathcal{S}, c)$, *onde $n := |E|$.*

Demonstração: A prova de que o algoritmo é uma H_n-aproximação é por indução em $|E|$. Se $|E| = 0$, então o algoritmo devolve o conjunto vazio, que é uma cobertura de custo mínimo. Agora, suponha que $|E| > 0$, e portanto $|\mathcal{S}| > 0$. Adote as abreviaturas $n := |E|$ e $k := |Z \cap E|$, onde Z é o conjunto escolhido na linha 3 do algoritmo. Como $|E'| = n - k < |E|$, podemos supor que a coleção $\mathcal{T}'$ produzida na linha 7 do algoritmo é uma cobertura de E' e que $c'(\mathcal{T}') \leq H_{n-k} \operatorname{opt}(E', \mathcal{S}', c')$. Além disso, $\operatorname{opt}(E', \mathcal{S}', c') \leq \operatorname{opt}(E, \mathcal{S}, c)$, uma vez que toda cobertura de E contém uma cobertura de E' formada por conjuntos de $\mathcal{S}'$. Com isso, em virtude de (2.2),

$$\begin{aligned}
&c(\{Z\} \cup \mathcal{T}') = \\
&\quad = c_Z + c'(\mathcal{T}') \\
&\quad \leq \tfrac{k}{n} \operatorname{opt}(E, \mathcal{S}, c) + H_{n-k} \operatorname{opt}(E, \mathcal{S}, c) \\
&\quad \leq \Big(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n-k+1} + H_{n-k}\Big) \operatorname{opt}(E, \mathcal{S}, c) \\
&\quad = H_n \operatorname{opt}(E, \mathcal{S}, c)\,.
\end{aligned}$$

Ademais, o algoritmo é polinomial, já que consome tempo $O(|E||\mathcal{S}|)$. □

O algoritmo MINCC-CHVÁTAL pode produzir coberturas de custo arbitrariamente próximo de $H_n \operatorname{opt}(E, \mathcal{S}, c)$, onde $n := |E|$. Mais precisamente, para cada ε positivo, existe uma instância do problema para a qual o algoritmo produz uma cobertura de custo $H_n \operatorname{opt}(E, \mathcal{S}, c)/(1+\varepsilon)$: basta tomar $E := \{1, \ldots, n\}$, $\mathcal{S} := \{E, \{1\}, \ldots, \{n\}\}$, $c_E := 1+\varepsilon$ e $c_{\{i\}} := 1/i$ para cada i. Uma cobertura de custo mínimo é $\{E\}$ e o custo desta cobertura é $1+\varepsilon$. Por outro lado, o algoritmo MINCC-CHVÁTAL produz a cobertura $\{\{1\}, \ldots, \{n\}\}$, cujo custo é H_n.

Assintoticamente, o algoritmo MINCC-CHVÁTAL tem a melhor razão possível para o problema MINCC, pois sabe-se [RS97] que $H_n \leq 1 + \ln n$ e existe uma constante positiva α para a qual não existe algoritmo com razão de aproximação menor que $\alpha \ln n$, com $n = |E|$, para o MINCC $(E, \mathcal{S}, c)$, a menos que P = NP. Veremos mais sobre isso no capítulo 8.

2.3 Mochila

Nesta seção tratamos de um outro problema de otimização bem conhecido: o **problema da mochila** (*knapsack problem*). Esse problema tem importantes aplicações, como por exemplo o carregamento ótimo de *containers*. Podemos enunciá-lo da seguinte maneira.

> **Problema** MOCHILA (m, n, v, w): *Dados um número m em $\mathbb{Q}_{\geq}$, um número n em $\mathbb{Z}_{>}$, um número v_i em $\mathbb{Z}_{\geq}$ e um número w_i em $\mathbb{Q}_{\geq}$ para cada i em $\{1, \ldots, n\}$, encontrar um subconjunto S de $\{1, \ldots, n\}$ que maximize $v(S)$ sob a restrição $w(S) \leq m$.*

Os números v_i e w_i podem ser interpretados como valor e peso respectivamente de um objeto i. O número m pode ser interpretado como a capacidade de uma mochila, ou seja, o peso máximo que a mochila comporta. O objetivo do problema é então encontrar uma coleção de objetos a mais valiosa possível que respeite a capacidade da mochila.[1]

Uma abordagem de programação dinâmica [CLR92] resolve o problema MOCHILA: basta construir uma tabela W onde W_{ij} é o peso mínimo de um subconjunto de $\{1, \ldots, i\}$ cujo valor é pelo menos j:

$$W_{ij} := \min \{ w(S) : S \subseteq \{1, \ldots, i\} \text{ e } v(S) \geq j \} .$$

[1] Convém não confundir o problema MOCHILA com a sua variante fracionária [CLR92], que permite escolher qualquer fração de um objeto.

Aqui i varia de 0 a n e j de 0 ao valor ótimo $\mathrm{opt}(m, n, v, w)$ do problema mais 1. Se não há subconjunto de $\{1, \ldots, i\}$ de valor pelo menos j, então $W_{ij} = \infty$. Abaixo, segue o algoritmo e, na figura 2.2, um exemplo.

Algoritmo MOCHILA-EXATO (m, n, v, w)

```
1   para i de 0 a n faça W_{i0} ← 0
2   j ← 0
3   repita
4       j ← j + 1
5       W_{0j} ← ∞
6       para i de 1 a n faça
7           se v_i ≥ j
8               então W_{ij} ← min {W_{i-1,j} , w_i}
9               senão W_{ij} ← min {W_{i-1,j} , w_i + W_{i-1,j-v_i}}
10  até que W_{nj} > m
11  seja S um subconjunto de {1, ..., n}
12      com w(S) = W_{n,j-1} e v(S) ≥ j - 1
13  devolva S
```

W	0	1	2	3	4	5	6	7	8	9	10
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞	∞	∞	∞	∞	∞
2	0	2	4	6	∞	∞	∞	∞	∞	∞	∞
3	0	1	1	1	3	5	7	∞	∞	∞	∞
4	0	1	1	1	2	3	3	3	5	7	9
5	0	1	1	1	2	3	3	3	5	7	9

Figura 2.2: Aplicação do algoritmo MOCHILA-EXATO à instância $m = 7$, $n = 5$, $v = (2, 1, 3, 4, 1)$, $w = (4, 2, 1, 2, 2)$ do problema MOCHILA. A figura exibe a tabela W que o algoritmo calcula. O valor ótimo dessa instância é 9 e existem duas soluções ótimas: $\{1, 3, 4\}$ e $\{2, 3, 4, 5\}$.

As linhas 7 a 9 são fáceis de entender. Suponha que S é um conjunto de peso mínimo dentre os que estão incluídos em $\{1, \ldots, i\}$ e têm valor pelo menos j. Se $i \notin S$ então S é um conjunto de peso mínimo dentre os que estão incluídos em $\{1, \ldots, i-1\}$ e têm valor pelo menos j. Se $i \in S$

então $S \setminus \{i\}$ é um conjunto de peso mínimo dentre os que estão incluídos em $\{1, \ldots, i-1\}$ e têm valor pelo menos $j - v_i$. (Se $v_i \geq j$ então $S = \{i\}$.)

Para justificar a condição de parada na linha 10, observe que $W_{nj} \leq W_{nk}$ para todo $k \geq j$, uma vez que todo subconjunto S de $\{1, \ldots, n\}$ que satisfaz $v(S) \geq k$ também satisfaz $v(S) \geq j$. Assim, se $W_{nj} > m$, então $W_{nk} > m$ para todo $k > j$ e, portanto, podemos interromper os cálculos.

Infelizmente, o número de execuções das linhas 3 a 10 pode não ser polinomial em $\langle v \rangle$. Por exemplo, se $n = 1$ e $m > v_1$, esse número é $v_1 + 1$ enquanto que $\langle v_1 \rangle = O(\log v_1)$. A bem da verdade, o problema MOCHILA é NP-difícil [GJ79]. Podemos dizer entretanto que o algoritmo MOCHILA-EXATO consome tempo proporcional ao número de componentes da matriz W. Esse número é limitado por $(n+1)(\sigma_v + 1)$, onde

$$\sigma_v := \textstyle\sum_{i\,:\,w_i \leq m} v_i \,,$$

já que o valor ótimo do problema MOCHILA (m, n, v, w) não ultrapassa σ_v (com $\sigma_v = 0$ se todo $w_i > m$). Não é difícil verificar que as linhas 11 a 12 podem ser executadas em tempo $O(n + \sigma_v)$. Assim o consumo total de tempo do algoritmo MOCHILA-EXATO é $O(n(\sigma_v + 1))$.

Esquema de aproximação

Seja ε um número racional no intervalo aberto $(0, 1)$. Vamos ver agora como usar o MOCHILA-EXATO para obter uma $(1-\varepsilon)$-aproximação polinomial para o problema MOCHILA. O seguinte algoritmo, proposto por Ibarra e Kim [IK75], faz uma mudança de escala nos valores de uma instância (m, n, v, w) do problema, obtendo uma outra instância para a qual o algoritmo MOCHILA-EXATO consome tempo polinomial em $m, n, \langle v \rangle$ e $\langle w \rangle$.

Algoritmo MOCHILA-IK$_\varepsilon$ (m, n, v, w)

se $w_i > m$ para todo i
    então devolva $\emptyset$
ϑ senão $\vartheta \leftarrow \max_{i\,:\,w_i \leq m} v_i$
λ $\lambda \leftarrow \varepsilon\vartheta / n$
        para i de 1 a n faça $u_i \leftarrow \lfloor v_i / \lambda \rfloor$
        $S \leftarrow$ MOCHILA-EXATO (m, n, u, w)
        devolva S

Na linha 5 do algoritmo, $\lfloor x \rfloor$ é o maior inteiro que não excede x. $\lfloor x \rfloor$

Como S é uma solução ótima do problema MOCHILA (m, n, u, w), temos que $\sum_{i \in S} w_i \leq m$, ou seja, S é uma solução viável do problema MOCHILA (m, n, v, w).

O valor do objeto mais valioso cujo peso não excede a capacidade da mochila, que é o número ϑ, é uma delimitação inferior para o valor ótimo do problema:

$$\operatorname{opt}(m, n, v, w) \geq \vartheta \,. \tag{2.3}$$

Essa delimitação é usada na análise do algoritmo.

Teorema 2.3: *O algoritmo* MOCHILA-IK$_\varepsilon$ *é uma* $(1-\varepsilon)$*-aproximação polinomial para o problema* MOCHILA.

Demonstração: O conjunto S na linha 6 do algoritmo é uma solução ótima do problema MOCHILA(m, n, u, w). Se S^* é uma solução ótima do MOCHILA(m, n, v, w), então

$$\begin{aligned}
\textstyle\sum_{i \in S} v_i &\geq \lambda \textstyle\sum_{i \in S} u_i \\
&\geq \lambda \textstyle\sum_{i \in S^*} u_i && (2.4)\\
&\geq \lambda \textstyle\sum_{i \in S^*} (\frac{v_i}{\lambda} - 1) && (2.5)\\
&= v(S^*) - \lambda\,|S^*| \\
&\geq v(S^*) - \lambda\, n \\
&= v(S^*) - \varepsilon\vartheta \\
&\geq (1 - \varepsilon) \operatorname{opt}(m, n, v, w) \,. && (2.6)
\end{aligned}$$

A desigualdade (2.4) vale pois S é uma solução ótima do problema MOCHILA(m, n, u, w). Já (2.5) vale pois $u_i > v_i/\lambda - 1$ para todo i em S^*. Finalmente, (2.6) vale por (2.3).

Quanto ao tempo de execução do algoritmo, temos o seguinte. A linha 6 consome tempo $O(n(\sigma_u + 1))$, onde $\sigma_u := \sum_{i\,:\,w_i \leq m} u_i$. Mas $u_i \leq v_i/\lambda \leq n/\varepsilon$, para todo i tal que $w_i \leq m$. Assim $\sigma_u \leq n^2/\varepsilon$ e portanto o MOCHILA-IK$_\varepsilon$ consome tempo $O(n^3/\varepsilon)$. □

Note que o algoritmo MOCHILA-IK$_\varepsilon$ é um esquema que nos fornece, para cada ε racional no intervalo $(0, 1)$, uma $(1-\varepsilon)$-aproximação que consome tempo polinomial em n/ε para resolver a instância MOCHILA (m, n, v, w). Assim, MOCHILA-IK$_\varepsilon$ é conhecido um esquema de aproximação plenamente polinomial (*fully polynomial-time approximation scheme*) (capítulo 8).

2.4 Caixeiro viajante

Nesta seção, abordamos um problema que surge em várias aplicações práticas, como a perfuração de placas de circuito impresso e a determinação de rotas de transporte de custo mínimo. Informalmente, o problema consiste em determinar uma rota de comprimento mínimo que passe exatamente uma vez em cada um dos pontos de um conjunto dado.

É conveniente formalizar o problema na linguagem de teoria dos grafos (apêndice A). Em particular, convém recorrer ao conceito de circuito hamiltoniano: um circuito que contém todos os vértices do grafo. O **problema do caixeiro viajante** (*traveling salesman problem*), denotado por TSP, é definido da seguinte maneira:

> **Problema** TSP (G, c): *Dados um grafo G e um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e, determinar um circuito hamiltoniano C que minimize $c(C)$.*

Esse é talvez o mais famoso problema de otimização combinatória, em parte graças às conexões com vários outros problemas de otimização. Ele é NP-difícil mesmo se $c_e \in \{1, 2\}$ para toda aresta e [GJ79]. Além disso, não se conhece um algoritmo de aproximação com razão constante para o problema, conforme veremos no capítulo 8. Nesta seção, nos restringimos a um caso particular do TSP que admite algoritmo de aproximação com razão constante.

Caixeiro viajante métrico

Suponha que o grafo G é completo e temos um custo c_{ij} associado a cada par ij de vértices. Dizemos que os custos satisfazem a **desigualdade triangular** se

$$c_{ik} \;\leq\; c_{ij} + c_{jk} \tag{2.7}$$

para quaisquer três vértices i, j e k. O TSP restrito ao conjunto de instâncias (G, c) em que G é completo e c satisfaz a desigualdade triangular é conhecido como **problema do caixeiro viajante métrico** e

TSPM será denotado aqui por TSPM. O problema é NP-difícil [GJ79].

Discutimos a seguir dois algoritmos de aproximação para o TSPM. A estratégia utilizada pelos dois algoritmos tem quatro passos: (1) construir uma árvore geradora T de G; (2) acrescentar novas arestas a T para obter um novo grafo T' cujos vértices têm grau par; (3) obter um ciclo euleriano P em T'; e (4) obter um circuito hamiltoniano em G a partir de P. A diferença entre os dois algoritmos está apenas na política adotada para acrescentar novas arestas à árvore T.

Chamamos o número $c(S)$ de **custo** de S, onde S pode ser um circuito, um ciclo, uma árvore, um caminho ou um conjunto de arestas.

O passo 1 da estratégia envolve uma árvore geradora de custo mínimo (*minimum-cost spanning tree*). Uma árvore geradora de custo mínimo dá uma boa delimitação inferior para o valor ótimo do problema TSPM (G, c): se removemos uma aresta de um circuito hamiltoniano temos uma árvore geradora de custo não superior ao do circuito. Portanto,

$$\mathrm{opt}(G, c) \geq c(T) \ . \tag{2.8}$$

Existem algoritmos simples e eficientes [BM76, CLR92] para construir uma árvore geradora de custo mínimo em um grafo conexo. Vamos designar por MST um algoritmo qualquer desse tipo. O consumo de tempo do algoritmo é $O(n^2)$, onde n é o número de vértices do grafo.

MST

A operação a que se refere o passo 2 pode ser formalizada da seguinte maneira. Para qualquer conjunto F de pares não-ordenados de vértices de T, seja $T + F$ o multigrafo $(V_T, E_T \dot{\cup} F)$, onde $E_T \dot{\cup} F$ denota o multiconjunto que tem duas cópias de cada elemento de $E_T \cap F$ (apêndice A). Como o grafo subjacente é completo, cada aresta do multigrafo $T + F$ tem um custo bem definido.

$T + F$

Um **ciclo euleriano** em um grafo ou multigrafo T' é qualquer ciclo que contém todas as arestas de T'. Um multigrafo conexo T' tem um ciclo euleriano se e somente se cada um de seus vértices tem grau par [BM76]. São bem conhecidos os algoritmos [AHU74] que constroem um ciclo euleriano em um multigrafo conexo sem vértices de grau ímpar. Para os propósitos do passo 3, vamos designar por Euler um algoritmo qualquer desse tipo. O consumo de tempo do algoritmo é proporcional ao número de arestas do multigrafo.

Euler

O passo 4 da estratégia transforma um ciclo gerador, ou seja, um ciclo que contém todos os vértices do multigrafo, em um circuito hamiltoniano. O procedimento é simples: basta extrair uma subseqüência maximal sem vértices repetidos da seqüência $(v_0, v_1, \ldots, v_m)$ de vértices do ciclo gerador. Isso pode ser realizado pelo seguinte procedimento:

Atalho

$w_0 \leftarrow v_0$
$n \leftarrow 0$
para i de 1 a m faça
    se $v_i \notin \{w_0, \ldots, w_n\}$
        então $n \leftarrow n + 1$
                $w_n \leftarrow v_i$

Como o grafo é completo, a seqüência $(w_0, w_1, \ldots, w_n, w_0)$ define um circuito. O circuito contém todos os vértices do grafo, pois o ciclo dado contém todos os vértices. Cada par (w_j, w_{j+1}) de vértices consecutivos no circuito é ligado por um segmento $(v_i, v_{i+1}, \ldots, v_{i+p})$ do ciclo. Graças à desigualdade triangular (2.7), o custo da aresta $w_j w_{j+1}$ não é maior que o custo do segmento. Portanto, o custo do circuito resultante C não é maior que o do ciclo dado P:

$$c(C) \leq c(P)\ . \tag{2.9}$$

Denotamos por ATALHO (*short-cut*) o procedimento que acabamos de descrever. O tempo gasto pelo procedimento é proporcional ao número de arestas do ciclo dado e portanto ao número de arestas do grafo.

Algoritmo de Rosenkrantz, Stearns e Lewis

No algoritmo descrito a seguir, o multigrafo T' (passo 2 da estratégia) é obtido por meio da duplicação de cada uma das arestas da árvore geradora T. O algoritmo aparece em um artigo de Rosenkrantz, Stearns e Lewis [RSL77].

Algoritmo TSPM-RSL (G, c)

```
1  T ← MST (G, c)
2  T' ← T + E_T
3  P ← EULER (T')
4  C ← ATALHO (P)
5  devolva C
```

Evidentemente, todo vértice de T' tem grau par e, portanto, T' tem um ciclo euleriano. O algoritmo EULER determina um tal ciclo. Como o conjunto de vértices de T' é V_G, o ciclo euleriano P é gerador. O circuito C devolvido por ATALHO na linha 4 é, então, um circuito hamiltoniano de G.

Teorema 2.4: *O algoritmo* TSPM-RSL *é uma* 2*-aproximação polinomial para o* TSPM.

Demonstração: Como P é um ciclo euleriano em $T + E_T$, temos que $c(P) = 2\,c(T)$. Então, por (2.8) e (2.9),

$$c(C) \ \leq\ c(P) \ =\ 2\,c(T) \ \leq\ 2\,\mathrm{opt}(G, c)\ .$$

A linha 1 do algoritmo consome tempo $O(|V_G|^2)$. As demais linhas consomem tempo $O(|V_G|)$, pois o número de arestas de T' é menor que $2|V_G|$. Em suma, o algoritmo é polinomial. □

Algoritmo de Christofides

Um **emparelhamento** (*matching*) em um grafo é um conjunto de arestas sem extremos em comum (ou seja, cada vértice pertence a no máximo uma das arestas do emparelhamento). Um emparelhamento M é **perfeito** se cada vértice do grafo pertence a uma aresta de M. O algoritmo de Edmonds [LP86], que denotaremos por EDMONDS, encontra um emparelhamento perfeito de custo mínimo em tempo $O(n^3)$, onde n é o número de vértices do grafo. EDMONDS

O algoritmo de Christofides [Chr76] acrescenta à árvore geradora T um emparelhamento perfeito no subgrafo de G induzido pelos vértices que têm grau ímpar em T.

Algoritmo TSPM-CHRISTOFIDES (G, c)
1 $T \leftarrow$ MST (G, c)
2 seja I o conjunto dos vértices de grau ímpar de T
3 $M \leftarrow$ EDMONDS $(G[I], c)$
4 $T' \leftarrow T + M$
5 $P \leftarrow$ EULER (T')
6 $C \leftarrow$ ATALHO (P)
7 devolva C

Como M é um emparelhamento perfeito em $G[I]$, todo vértice de $T + M$ tem grau par e, portanto, o multigrafo T' na linha 4 tem um ciclo euleriano. O ciclo é gerador pois T é geradora. Na linha 6 do algoritmo, C é um circuito hamiltoniano de G.

Teorema 2.5: *O algoritmo* TSPM-CHRISTOFIDES *é uma* $\frac{3}{2}$-*aproximação polinomial para o* TSPM.

Demonstração: Precisamos mostrar que C tem custo no máximo $\frac{3}{2}\,\mathrm{opt}(G, c)$. De acordo com (2.9), temos que $c(C) \leq c(P)$. Por outro lado, $c(P) = c(T') = c(T) + c(M)$. Usando (2.8), temos que $c(C) \leq c(T) + c(M) \leq \mathrm{opt}(G, c) + c(M)$. Portanto, para concluir que

$c(C) \leq \frac{3}{2}\,\mathrm{opt}(G,c)$, basta mostrar que

$$\mathrm{opt}(G,c) \geq 2\,c(M)\ . \tag{2.10}$$

Seja C^* uma solução ótima para o TSPM. Sejam $u_1, u_2, \ldots, u_{2k}$ os vértices de I na ordem em que aparecem em C^*. Como G é completo, a seqüência $D := (u_1, u_2, \ldots, u_{2k}, u_1)$ é um circuito em $G[I]$. Em outras palavras, D pode ser obtido de C^* pela substituição de cada segmento de C^* que liga u_i a u_{i+1} pela aresta $u_i u_{i+1}$ de G. A desigualdade triangular (2.7) garante que $c(D) \leq c(C^*)$. Além disso, como D tem comprimento par, E_D é a união de dois emparelhamentos perfeitos em $G[I]$ mutuamente disjuntos, digamos M' e M''. Logo,

$$2\,c(M) \leq c(M') + c(M'') = c(D) \leq c(C^*) = \mathrm{opt}(G,c)\ ,$$

sendo que a primeira desigualdade vale porque M é um emparelhamento perfeito de custo mínimo. Isso completa a prova de (2.10).

A linha 3 consome tempo $O(|V_G|^3)$, enquanto que as demais linhas consomem tempo $O(|V_G|^2)$. Portanto, o algoritmo é polinomial. $\square$

Proposto em 1976, TSPM-CHRISTOFIDES é ainda o melhor algoritmo de aproximação conhecido para o TSPM. O algoritmo TSPM-RSL pode ser uma boa alternativa em certas circunstâncias: ele consome menos tempo que o TSPM-CHRISTOFIDES e é bem mais simples, pois não envolve a determinação de um emparelhamento perfeito de custo mínimo.

Exercícios

2.1 Mostre que o algoritmo ESCALONAMENTO-GRAHAM (m, n, t) tem razão de aproximação $2 - \frac{1}{m}$. Para cada m, exiba uma instância para a qual o algoritmo produz um escalonamento que atinge tal razão em relação ao ótimo.

2.2 Considere a variante do algoritmo de Graham que coloca as tarefas em ordem não-decrescente de tempo antes de começar o processo de escalonamento. Mostre que essa variante é uma 4/3-aproximação polinomial para o ESCALONAMENTO. Exiba uma instância (m, n, t) para a qual o algoritmo produz um escalonamento de custo $\frac{4}{3}\,\mathrm{opt}(m, n, t)$.

2.3 Escreva uma versão iterativa do algoritmo MINCC-CHVÁTAL.

2.4 O **problema da cobertura máxima** (*maximum coverage problem*) consiste no seguinte:

> **Problema** MAXCC $(E, w, \mathcal{S}, k)$: *Dados um conjunto finito E, um peso w_e em $\mathbb{Q}_{\geq}$ para cada e em E, uma coleção $\mathcal{S}$ de subconjuntos de E e um inteiro não-negativo $k \leq |\mathcal{S}|$, encontrar uma subcoleção $\mathcal{T}$ de $\mathcal{B}$ tal que $|\mathcal{T}| = k$ e $w(\bigcup \mathcal{T})$ é máximo.*

Este problema é NP-difícil [GJ79]. Mostre que o algoritmo guloso que escolhe a cada passo um conjunto que cobre o subconjunto mais pesado de elementos descobertos é uma $(1 - (1{-}k^{-1})^k)$-aproximação polinomial para o problema. Lembre-se de que $1 - (1{-}k^{-1})^k > 1 - e^{-1}$, onde e é a base dos logaritmos naturais, e conclua que esse algoritmo é uma 0,63-aproximação para o MAXCC.

2.5 Construa instâncias do MINCC com custos unitários, ou seja, instâncias $(E, \mathcal{S}, c)$ com $c_S = 1$ para todo S em $\mathcal{S}$, para as quais o custo da cobertura produzida pelo algoritmo MINCC-CHVÁTAL pode chegar arbitrariamente perto de $H_n \operatorname{opt}(E, \mathcal{S}, c)$, onde $n := |E|$.

2.6 Lembre-se que $\ln x$ é a primitiva da função $1/x$. Deduza daí que $H_n \leq 1 + \ln n$. Conclua que o algoritmo MINCC-CHVÁTAL é uma $O(\log n)$-aproximação polinomial para o MINCC $(E, \mathcal{S}, c)$, onde $n := |E| > 1$.

2.7 Reescreva o algoritmo MOCHILA-EXATO (m, n, v, w) de modo que fique mais evidente que as linhas 11 e 12 do algoritmo podem ser executadas em tempo $O(n + \sigma_v)$, onde $\sigma_v := \sum_{i\,:\,w_i \leq m} v_i$.

2.8 Considere o seguinte algoritmo guloso para o problema da mochila: selecione um a um os objetos j cujo valor de v_j/w_j é máximo até que a capacidade da mochila seja atingida ou todos os objetos já foram considerados. Mostre que este algoritmo não tem razão de aproximação constante, ou seja, exiba uma família de instâncias (m, n, v, w) do problema para as quais a razão entre o valor da solução produzida pelo algoritmo e $\operatorname{opt}(m, n, v, w)$ é arbitrariamente pequena.

2.9 O **problema do empacotamento unidimensional** (*bin packing problem*) consiste no seguinte:

> **Problema** EMPACOTAMENTO (n, c): *Dados um inteiro positivo n e, para cada i em $\{1, \dots, n\}$, um número racional c_i no intervalo fechado $[0, 1]$, encontrar uma partição $\mathcal{B}$ de*

$\{1, \ldots, n\}$ *tal que* $c(B) \leq 1$ *para todo* B *em* $\mathcal{B}$ *e* $|\mathcal{B}|$ *seja mínimo.*

Este problema é NP-difícil no sentido forte [GJ79]. Mostre que o algoritmo abaixo é uma 2-aproximação polinomial para o EMPACOTAMENTO. Exiba uma instância (n, c) para a qual o algoritmo produz uma partição $\mathcal{B}$ tal que $|\mathcal{B}| = 2\,\mathrm{opt}(n, c)$.

Algoritmo EMPACOTAMENTO-NEXT-FIT (n, c)
$k \leftarrow 1$
$B_k \leftarrow \emptyset$
para i de 1 a n faça
    se $c_i \leq 1 - c(B_k)$
        então $B_k \leftarrow B_k \cup \{i\}$
        senão $k \leftarrow k + 1$
                $B_k \leftarrow \{i\}$
devolva $\{B_1, \ldots, B_k\}$

2.10 Construa uma família de instâncias (G, c) do TSPM para as quais o custo do circuito hamiltoniano obtido pelo algoritmo TSPM-RSL pode ser arbitrariamente próximo de $2\,\mathrm{opt}(G, c)$. Construa uma família de instâncias (G, c) do TSPM para as quais o custo do circuito hamiltoniano obtido pelo algoritmo de Christofides pode ser arbitrariamente próximo de $\frac{3}{2}\,\mathrm{opt}(G, c)$.

2.11 Considere as três seguintes variantes do TSPM. Na primeira, buscamos um caminho hamiltoniano de custo mínimo no grafo dado. Na segunda, queremos um caminho hamiltoniano de custo mínimo dentre os que começam em um dado vértice s. Na terceira, queremos um caminho hamiltoniano de custo mínimo dentre os que começam em um dado vértice s e terminam em um dado vértice t. Modifique o algoritmo de Christofides para que ele seja um algoritmo de aproximação com razão menor que 2 para cada uma destas variantes.

2.12 Mostre que os algoritmos TSPM-RSL e TSPM-CHRISTOFIDES podem produzir péssimos resultados se aplicados a instâncias do TSP que não satisfazem a desigualdade triangular.

Notas bibliográficas

A primeira versão desse capítulo teve por base o livro editado por Hochbaum [Hoc97], o livro de Cormen *et al.* [CLR92], o texto de Guimarães [Gui98], as notas de aula de Williamson [Wil98] e o relatório técnico de Christofides [Chr76]. O exercício 2.4 foi extraído do livro de Vazirani [Vaz01].

Para o problema do escalonamento em máquinas idênticas, Hochbaum e Shmoys [HS88] obtiveram um esquema de aproximação polinomial, ou seja, uma $(1+\varepsilon)$-aproximação polinomial para todo número racional positivo ε fixo. O algoritmo descrito no exercício 2.2 foi proposto por Graham [Gra69]. O algoritmo apresentado na seção 2.1 é conhecido como *list scheduling*. Diversas variantes do problema são discutidas na literatura. Uma boa amostra dessas variantes e suas aplicações encontra-se no livro de Brucker [Bru98] e no livro editado por Chrétienne *et al.* [CJLL95].

O problema da cobertura mínima por conjuntos, no caso especial em que todos os custos são unitários, foi estudado primeiramente por Johnson [Joh74], que apresentou uma $(1 + \ln n)$-aproximação, onde n é o número de elementos no conjunto que se quer cobrir. Lovász [Lov75], estudando coberturas em hipergrafos, obteve esse mesmo resultado. O algoritmo MINCC-CHVÁTAL aplicado a essas instâncias especiais coincide com esse algoritmo.

Um esquema de aproximação plenamente polinomial para o problema da mochila, que consome menos tempo e espaço do que os anteriormente conhecidos, foi recentemente obtido por Kellerer [KP99]. O problema da mochila tem diversas variantes (com precedência, não-linear, estocástica, multidimensional). O livro de Martello e Toth [MT90] é uma excelente resenha do estado-da-arte para muitas dessas variantes. Entre outras coisas, o livro apresenta resultados computacionais sobre implementações de vários algoritmos exatos e de aproximação e contém um disquete com os códigos das implementações.

O TSP é considerado um problema central na área de otimização combinatória. O livro editado por Lawler *et al.* [LLRS90], com contribuições de diversos autores, fornece uma resenha unificada, completa e atualizada até 1985. Uma resenha mais recente, devida a Jünger, Reinelt e Rinaldi [JRR95], discute várias técnicas, concentrando-se naquelas que têm se mostrado mais eficazes para resolver instâncias do TSP que surgem na prática. Dentre essas, destacamos a heurística de Lin e Kernighan [LK73] e os métodos *branch-and-cut*, que baseiam-se em um poliedro associado ao TSP [ABCC98].

A biblioteca TSPLIB, mantida por Reinelt [Rei00], é uma coletânea de instâncias do TSP de diversos tamanhos e tipos. Ela é usada em estudos comparativos da eficiência e eficácia de novas heurísticas e algoritmos de aproximação para o TSP.

Um esquema de aproximação polinomial para a versão euclidiana do TSP em dimensão fixa foi obtido por Arora [Aro98] e, logo em seguida, independentemente, por Mitchell [Mit99]. Um algoritmo de aproximação para o problema do caminho hamiltoniano de custo mínimo, nos moldes do algoritmo de Christofides, foi projetado por Hoogeveen [Hoo91] (exercício 2.11). Arora *et al.* [AGK$^+$98] apresentaram um esquema de aproximação polinomial para o TSPM restrito a grafos planares.

O sofisticado algoritmo de Edmonds [Edm65a, Edm65b, LP86] para encontrar um emparelhamento perfeito de custo mínimo é um marco na teoria algorítmica dos grafos e está ligado às origens da teoria de complexidade (apêndice E).

Capítulo 3

Método Primal

Vimos no capítulo anterior que técnicas bem conhecidas de projeto de algoritmos, como a programação dinâmica e o método guloso, podem resultar em bons algoritmos de aproximação. Neste capítulo e nos próximos três, mostramos como programação linear (apêndice C) pode ser usada na criação de bons algoritmos de aproximação.

Programação linear é uma ferramenta útil pois existem bons algoritmos para resolver programas lineares. Além disso, ela pode fornecer boas delimitações para o valor ótimo do problema em questão e, como já vimos, tais delimitações são essenciais para a análise da qualidade da solução produzida por um algoritmo de aproximação.

Para criar um algoritmo de aproximação baseado em programação linear, necessitamos de um programa linear que seja uma *relaxação* do problema de otimização: cada solução viável do problema combinatório deve corresponder a uma solução viável do programa linear.

3.1 Técnica do arredondamento

Uma técnica simples mas às vezes eficaz é a do **arredondamento**. Esta técnica consiste em arredondar uma solução ótima de um programa linear que represente o problema original. A técnica pode dar bons resultados não só com programas lineares, como veremos no capítulo 7. Nesta seção, vamos aplicar a técnica do arredondamento ao problema da cobertura mínima por conjuntos, já discutido na seção 2.2.

Problema MINCC $(E, \mathcal{S}, c)$: *Dados um conjunto finito E, uma coleção finita $\mathcal{S}$ de conjuntos finitos que cobre E e um custo c_S em $\mathbb{Q}_{\geq}$ para cada S em $\mathcal{S}$, encontrar uma cobertura $\mathcal{T}$ de E que minimize $c(\mathcal{T})$.*

Para aplicar a técnica do arredondamento, precisamos formular um programa linear que represente o MINCC. Considere o seguinte programa linear: encontrar um vetor x indexado por $\mathcal{S}$ que

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & \sum_{S\,:\,e\in S} x_S & \geq & 1 & \text{para cada } e \text{ em } E\,, \\ & x_S & \geq & 0 & \text{para cada } S \text{ em } \mathcal{S}\,. \end{array} \tag{3.1}$$

Neste programa, tem-se uma variável x_S para cada S em $\mathcal{S}$ e, para cada e em E, uma restrição que corresponde à exigência de que e seja coberto. O programa linear (3.1) é limitado, pois $cx \geq 0$ para qualquer solução viável x de (3.1), e é viável já que o vetor característico de $\mathcal{S}$ satisfaz as restrições de (3.1). Assim, pelo teorema forte da dualidade (teorema C.3, apêndice C), o programa (3.1) tem uma solução ótima racional. Como o vetor característico de qualquer cobertura de E é viável em (3.1), temos a seguinte delimitação inferior para o valor ótimo do problema MINCC $(E, \mathcal{S}, c)$:

$$\text{opt}(E, \mathcal{S}, c) \geq c\hat{x} \tag{3.2}$$

para toda solução ótima $\hat{x}$ de (3.1).

Se todos os componentes de uma solução ótima $\hat{x}$ são inteiros, a subcoleção $\{S \in \mathcal{S} : \hat{x}_S > 0\}$ é uma solução ótima do MINCC, ou seja, é uma cobertura de E de custo mínimo. Em geral, porém, vários componentes de $\hat{x}$ são fracionários. Ao arredondarmos os valores desses componentes de maneira apropriada, podemos obter uma cobertura de E. A seguir, analisamos uma possível maneira de realizar este arredondamento.

Para cada e em E, seja f_e a freqüência de e em $\mathcal{S}$, isto é, o número
β de elementos de $\mathcal{S}$ aos quais e pertence. Seja β a maior das freqüências. Como $\mathcal{S}$ cobre E, temos que $\beta > 0$. O próximo algoritmo, projetado por Hochbaum [Hoc82], escolhe um conjunto S em $\mathcal{S}$ para fazer parte da cobertura se e somente se o valor da variável $\hat{x}_S$ é pelo menos $1/\beta$.

Algoritmo MINCC-HOCHBAUM $(E, \mathcal{S}, c)$

1 seja $\hat{x}$ uma solução ótima racional de (3.1)
2 para cada e em E faça $f_e \leftarrow |\{S \in \mathcal{S} : e \in S\}|$
3 $\beta \leftarrow \max_{e \in E} f_e$
4 $\mathcal{T} \leftarrow \{S \in \mathcal{S} : \hat{x}_S \geq 1/\beta\}$
5 devolva $\mathcal{T}$

A coleção $\mathcal{T}$ é uma cobertura, como passamos a mostrar. Cada e em E pertence a no máximo β conjuntos de $\mathcal{S}$. Este fato, juntamente com a restrição $\sum_{S\,:\,e \in S} x_S \geq 1$, garante que $\hat{x}_S \geq 1/\beta$ para algum S que contém e. Portanto, $\mathcal{T}$ contém algum S que contém e, ou seja, $\mathcal{T}$ é uma cobertura de E.

Teorema 3.1: *O algoritmo* MINCC-HOCHBAUM *é uma β-aproximação polinomial para o* MINCC $(E, \mathcal{S}, c)$, *onde β é o número máximo de vezes que um elemento de E aparece em conjuntos de $\mathcal{S}$.*

Demonstração: O custo da cobertura $\mathcal{T}$ produzida pelo algoritmo é

$$c(\mathcal{T}) \;=\; \sum_{S \in \mathcal{T}} c_S \;\leq\; \beta \sum_{S \in \mathcal{T}} c_S \hat{x}_S \;\leq\; \beta\, c\hat{x} \;\leq\; \beta \operatorname{opt}(E, \mathcal{S}, c)\,,$$

onde a primeira desigualdade vale pois $\beta\, \hat{x}_S \geq 1$ para todo S em $\mathcal{T}$ e a última vale por (3.2).

O programa linear (3.1) tem $|\mathcal{S}|$ variáveis e $|E|$ restrições (as restrições $x_S \geq 0$ são implícitas) e, portanto, o tamanho do sistema (3.1) é $(|\mathcal{S}| + 1)|E| + \langle c \rangle$. Como programas lineares podem ser resolvidos em tempo polinomial (fato C.4), a linha 1 do algoritmo pode ser executada em tempo polinomial. As demais linhas claramente também podem ser executadas em tempo polinomial em $|E|$ e $|\mathcal{S}|$. Assim, o algoritmo é polinomial. $\square$

3.2 Técnica métrica

Como na seção anterior, usaremos uma solução ótima $\hat{x}$ de um programa linear associado ao problema combinatório. Desta vez, $\hat{x}$ é um vetor indexado pelas arestas de um grafo e cada componente $\hat{x}_e$ será interpretado como o comprimento da aresta e.

Vamos descrever a aplicação dessa técnica ao problema do multicorte mínimo. Dados um grafo G e um conjunto K de pares de vértices, dizemos que um caminho de s a t é um K-**caminho** se $\{s,t\} \in K$. Um conjunto M de arestas é um K-**multicorte** se não existe K-caminho no grafo $G-M$. O **problema do multicorte mínimo** (*minimum multicut problem*) consiste no seguinte:

> **Problema** MINMCUT (G, K, c): *Dados um grafo G, um conjunto K de pares de vértices e um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e, encontrar um K-multicorte M que minimize $c(M)$.*

Há um bem conhecido algoritmo polinomial [AMO93] para o caso em que $|K| = 1$. Para o caso em que $|K| = 2$, também existe um algoritmo polinomial [Hu63, Ita78]. Dahlhaus *et al.* [DJP$^+$94] mostraram que o problema é NP-difícil mesmo quando restrito às instâncias em que $|K| = 3$.

$\mathcal{P}$ Denote por $\mathcal{P}$ o conjunto de todos os K-caminhos e considere o seguinte problema de programação linear: encontrar um vetor x indexado por E_G que

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & x(E_P) & \geq & 1 & \text{para cada } P \text{ em } \mathcal{P}\,, \\ & x_e & \geq & 0 & \text{para cada } e \text{ em } E_G\,. \end{array} \tag{3.3}$$

O vetor característico de E_G é viável em (3.3) e $cx \geq 0$ para qualquer solução viável x de (3.3). Portanto, de acordo com o teorema forte da dualidade, o programa (3.3) tem uma solução ótima racional. A relação entre este programa e o MINMCUT é simples: o vetor característico de qualquer K-multicorte é viável em (3.3) e portanto

$$\text{opt}(G, K, c) \geq c\hat{x} \tag{3.4}$$

para qualquer solução ótima $\hat{x}$ de (3.3).

O algoritmo abaixo foi concebido por Garg, Vazirani e Yannakakis [GVY96] e fornece a melhor razão de aproximação conhecida para o MINMCUT. Ele supõe que $K \neq \emptyset$; em caso contrário, o problema é trivial.

Algoritmo MINMCUT-GVY (G, K, c)

```
seja x̂ uma solução ótima racional de (3.3)
k ← |K|
M ← CENTRAL (G, k, K, c, x̂)
devolva M
```

k

O algoritmo Central, que detalhamos adiante, produz um K-multicorte M tal que

$$c(M) \;\leq\; (4 \ln 2k)\, cx \,. \tag{3.5}$$

Teorema 3.2: *O algoritmo* MinMCut-GVY *é uma* $(4 \ln 2k)$-*aproximação polinomial para o* MinMCut (G, K, c), *sendo* $k := |K| > 0$.

Demonstração: Em virtude de (3.5) e (3.4), o custo $c(M)$ do multicorte M que o algoritmo devolve satisfaz

$$c(M) \;\leq\; (4 \ln 2k)\, c\hat{x} \;\leq\; (4 \ln 2k)\, \mathrm{opt}(G, K, c) \,.$$

O número de restrições do programa (3.3) pode ser exponencial em $|V_G|$. Ainda assim, a linha 1 do algoritmo MinMCut-GVY pode ser executada em tempo polinomial em $\langle G \rangle + \langle c \rangle$, já que temos um algoritmo de separação (apêndice C) para (3.3): interprete x_e como comprimento da aresta e, calcule um caminho de comprimento mínimo de s a t para cada $\{s, t\}$ em K. Se algum destes caminhos, digamos P, tiver comprimento menor que 1, o vetor x não satisfaz a restrição de (3.3) correspondente a P. Do contrário, x satisfaz todas as restrições de (3.3). Esse algoritmo de separação consome tempo polinomial em $\langle G \rangle$ (use o algoritmo de Dijkstra [AMO93], por exemplo) e portanto a linha 1 do algoritmo pode ser executada em tempo polinomial em $\langle G \rangle + \langle c \rangle$.

A solução $\hat{x}$ é racional e $\langle \hat{x} \rangle$ é limitado por um polinômio em $\langle G \rangle$ (fato C.4). Como mostraremos ao analisar o algoritmo Central, a linha 3 consome tempo polinomial em $\langle G \rangle + \langle c \rangle + \langle \hat{x} \rangle$ e portanto polinomial em $\langle G \rangle + \langle c \rangle$. Assim, podemos concluir que o algoritmo MinMCut-GVY é polinomial. □

Algoritmo central

O algoritmo Central recebe um grafo G, um inteiro $k \geq 1$, um conjunto K de no máximo k pares de vértices, um vetor racional c e um vetor racional não-negativo x tal que $x(E_P) \geq 1$ para todo K-caminho P. Dado o caráter recursivo do algoritmo, é necessário refinar a especificação (3.5): o algoritmo Central produz um K-multicorte M tal que

$$c(M) \;\leq\; (2 \ln 2k)(1 + \tfrac{1}{k}|K|)\, cx \,. \tag{3.6}$$

Para obter um tal multicorte, Central calcula uma partição (S, T) de S
T

V_G que separe os dois vértices de algum par em K de modo a manter $c(\delta(S))$ razoavelmente pequeno. A partição (S,T) induz a partição A $(A, \delta(S), B)$ de E_G, onde A é o conjunto das arestas que têm ambos os B extremos em S e B é o conjunto das arestas que têm ambos os extremos em T. O algoritmo é então aplicado, recursivamente, ao grafo (V_G, B).

Para descrever o algoritmo CENTRAL, precisamos de alguma nota- $x(s,u)$ ção. Para quaisquer vértices s e u, seja $x(s,u) := \min_P x(E_P)$, onde o mínimo é tomado sobre todos os caminhos P de s a u (é claro que $x(s,u) = \infty$ se não existe caminho de s a u). Podemos interpretar $x(s,u)$ como a distância de s a u. Para qualquer número ρ, denotamos $V(s,\rho)$ por $V(s,\rho)$ o conjunto de todos os vértices à distância no máximo ρ de s, ou seja,

$$V(s,\rho) := \{v \in V_G : x(s,v) \leq \rho\} .$$

Imagine que as arestas do grafo são tubos, sendo x_e o comprimento e c_e a área da secção transversal do tubo e. Com essa interpretação em $\vartheta(s,\rho)$ mente, denote por $\vartheta(s,\rho)$ o volume da parte da tubulação que dista no máximo ρ de s:

$$\vartheta(s,\rho) := c_A x_A + \textstyle\sum_{uv \in \delta(S),\, u \in S} c_{uv}(\rho - x(s,u)) ,$$

onde $S := V(s,\rho)$ e c_A e x_A são as restrições de c e x, respectivamente, ao conjunto de arestas $A := E_{G[S]}$. É claro que $\vartheta(s,\rho) \leq cx$ qualquer que seja ρ.

Agora estamos prontos para descrever o algoritmo CENTRAL.

Algoritmo CENTRAL (G, k, K, c, x)

se $K = \emptyset$
    então devolva $\emptyset$
    senão se $cx = 0$
            então devolva $\{e \in E_G : x_e > 0\}$
            senão sejam s e t os extremos de um K-caminho
                  seja $v_1, \ldots, v_n$ uma ordenação dos vértices
                      tal que $x(s,v_1) \leq \cdots \leq x(s,v_n)$
p_i para i de 1 a n faça $p_i \leftarrow x(s,v_i)$
                  $j \leftarrow 1 + \max\{i : p_i = 0\}$
                  enquanto $\vartheta(s,p_j) > ((2k)^{2p_j} - 1)\frac{1}{k}cx$
                      faça $j \leftarrow j + 1$
                  $S \leftarrow V(s, p_{j-1})$

13 $T \leftarrow V_G \setminus S$
14 $B \leftarrow E_{G[T]}$
15 $G_B \leftarrow (V_G, B)$ G_B
16 $K_B \leftarrow K \setminus \{\{s', t'\} :$ S separa s' de $t'\}$
17 $M_B \leftarrow$ Central (G_B, k, K_B, c_B, x_B)
18 devolva $\delta(S) \cup M_B$

Na linha 16, dizemos que S **separa** s' de t' se S contém exatamente um de s' e t'. Na linha 17, os vetores c_B e x_B são as restrições de c e x a B, respectivamente.

No fim da linha 9, temos $2 \leq j \leq n$ pois $p_1 = x(s,s) = 0$ e $p_n \geq x(s,t) \geq 1$. Após as linhas 10 e 11, como $p_n \geq 1$ e $k \geq 1$,

$$((2k)^{2p_n} - 1)\tfrac{1}{k}cx \geq ((2k)^2 - 1)\tfrac{1}{k}cx > (2k-1)\tfrac{1}{k}cx \geq cx \geq \vartheta(s, p_n)\,.$$

Na linha 17, as condições de aplicabilidade do algoritmo Central estão satisfeitas: $|K_B| \leq |K| \leq k$, x_B é não-negativo e $x_B(E_P) \geq 1$ para todo K_B-caminho P em G_B.

Lema 3.3: *Ao fim da linha* 12 *do algoritmo* Central, *temos que*

$$c(\delta(S)) \leq (2 \ln 2k)\Big(c_A x_A + c_{\delta(S)} x_{\delta(S)} + \tfrac{1}{k}cx\Big)\,,$$

onde c_A e x_A são as restrições de c e x ao conjunto $A := E_{G[S]}$ enquanto $c_{\delta(S)}$ e $x_{\delta(S)}$ são as restrições de c e x a $\delta(S)$.

Demonstração: É preciso verificar que, após as linhas 10 e 11,

$$p_{j-1} < p_j\,, \tag{3.7}$$
$$\vartheta(s, p_{j-1}) \geq ((2k)^{2p_{j-1}} - 1)\tfrac{1}{k}cx \quad \text{e} \tag{3.8}$$
$$\vartheta(s, p_j) \leq ((2k)^{2p_j} - 1)\tfrac{1}{k}cx\,. \tag{3.9}$$

Como já vimos acima, $2 \leq j \leq n$ após as linhas 10 e 11. Suponha pois que $j = 2$. Então, em virtude da linha 9, temos que $p_{j-1} = p_1 = 0 < p_j$. Além disso, ainda supondo $j = 2$, temos que

$$\vartheta(s, p_{j-1}) = \vartheta(s, p_1) = \vartheta(s, 0) = 0 = ((2k)^0 - 1)\tfrac{1}{k}cx\,,$$

enquanto que $\vartheta(s, p_j) \leq ((2k)^{2p_j} - 1)\frac{1}{k}cx$ em virtude das linhas 10 e 11. Agora analisemos o caso em que $j > 2$. Claramente, (3.8) e (3.9) valem em virtude das linhas 10 e 11. Disso segue que $p_{j-1} \neq p_j$. Como $p_{j-1} \leq p_j$, pelas linhas 6 e 7, temos que (3.7) vale.

Agora prosseguimos com a prova do lema. De (3.8) e (3.9) temos que

$$\frac{\vartheta(s,p_j)+\frac{1}{k}cx}{\vartheta(s,p_{j-1})+\frac{1}{k}cx} \;\leq\; (2k)^{2(p_j-p_{j-1})}\,. \tag{3.10}$$

Tomando-se o logaritmo natural do lado esquerdo de (3.10) e desenvolvendo-o, obtemos

$$\begin{aligned}
&\ln(\vartheta(s,p_j)+\tfrac{1}{k}cx)-\ln(\vartheta(s,p_{j-1})+\tfrac{1}{k}cx) \;= \\
&\quad= \int_{p_{j-1}}^{p_j}\frac{d}{d\rho}\ln(\vartheta(s,\rho)+\tfrac{1}{k}cx)\,d\rho \\
&\quad= \int_{p_{j-1}}^{p_j}\frac{c(\delta(S))}{\vartheta(s,\rho)+\frac{1}{k}cx}\,d\rho\,,
\end{aligned}$$

pois a derivada de $\vartheta(s,\rho)+\frac{1}{k}cx$ em relação a ρ no intervalo (p_{j-1},p_j) é $c(\delta(S))$. Fazendo-se o mesmo com o lado direito de (3.10), obtemos

$$2(p_j-p_{j-1})\ln 2k \;=\; \int_{p_{j-1}}^{p_j}(2\ln 2k)\,d\rho\,.$$

Como o logaritmo é uma função crescente, concluímos de (3.10) que

$$\int_{p_{j-1}}^{p_j}\frac{c(\delta(S))}{\vartheta(s,\rho)+\frac{1}{k}cx}\,d\rho \;\leq\; \int_{p_{j-1}}^{p_j}(2\ln 2k)\,d\rho\,.$$

Então, para algum ρ no intervalo (p_{j-1},p_j), que não é vazio em virtude de (3.7), temos que $c(\delta(S))/(\vartheta(s,\rho)+\frac{1}{k}cx)\leq(2\ln 2k)$.

Para concluir o lema, resta mostrar que $\vartheta(s,\rho)\leq c_Ax_A+c_{\delta(S)}x_{\delta(S)}$. Para isso, note que, para todo uv em $\delta(S)$ com u em S, temos que $x(s,v)>\rho$ e $x(s,v)\leq x(s,u)+x_{uv}$, donde $\rho-x(s,u)<x_{uv}$. Como $S=V(s,p_{j-1})=V(s,\rho)$, segue que

$$\begin{aligned}
\vartheta(s,\rho) &= c_Ax_A+\textstyle\sum_{uv\in\delta(S),\,u\in S}c_{uv}(\rho-x(s,u)) \\
&\leq c_Ax_A+\textstyle\sum_{uv\in\delta(S)}c_{uv}x_{uv} \\
&= c_Ax_A+c_{\delta(S)}x_{\delta(S)}\,,
\end{aligned}$$

como queríamos demonstrar. □

Finalmente, estamos preparados para apresentar a análise do algoritmo CENTRAL.

Teorema 3.4: *O algoritmo* CENTRAL (G, k, K, c, x) *produz um K-multicorte M tal que*

$$c(M) \leq (2 \ln 2k)(1 + \tfrac{1}{k}|K|)cx \tag{3.11}$$

e consome tempo polinomial em $\langle G \rangle + \langle c \rangle + \langle x \rangle$.

Demonstração: Vamos começar provando que, após a execução das linhas 10 e 11,

$$p_{j-1} < \tfrac{1}{2}\,. \tag{3.12}$$

Para isso, seja h o menor natural tal que $p_h \geq \frac{1}{2}$. Tal h existe e é maior que 1 já que $p_n \geq x(s,t) \geq 1$ e $p_1 = 0$. Como $\vartheta(s, p_h) \leq cx$ e $(2k)^{2p_h} - 1 \geq k$, pois $k \geq 1$ e $p_h \geq \frac{1}{2}$, então $\vartheta(s, p_h) \leq ((2k)^{2p_h} - 1)\frac{1}{k}cx$. Assim, $j \leq h$ depois da execução da linha 11 e, como $p_{h-1} < \frac{1}{2}$ pela escolha de h e porque $p_1 = 0$, a desigualdade (3.12) vale.

Agora vamos verificar que o conjunto devolvido pelo algoritmo é de fato um K-multicorte. Se $K = \emptyset$, então o conjunto vazio que o algoritmo devolve é um multicorte. Se $cx = 0$, o conjunto $\{e \in E_G : x_e > 0\}$ é um K-multicorte: como $x(E_P) \geq 1$ para todo K-caminho P, todo K-caminho tem pelo menos uma aresta e com $x_e > 0$. Suponha agora que $K \neq \emptyset$ e $cx > 0$. Neste caso, o algoritmo devolve $M := \delta(S) \cup M_B$; podemos supor, por hipótese de indução, que M_B é um K_B-multicorte no grafo G_B. Vamos verificar que M é um K-multicorte em G. Para quaisquer dois vértices u e v em S,

$$x(u, v) \leq x(u, s) + x(s, v) < 1\,, \tag{3.13}$$

pois distâncias satisfazem a desigualdade triangular e vale (3.12). Suponha agora que P é um K-caminho em G. Como $x(E_P) \geq 1$, a desigualdade (3.13) garante que P tem pelo menos um extremo fora de S. Se P tem um vértice em S, então também tem pelo menos uma aresta em $\delta(S)$. Se P não tem vértices em S, então também é um K_B-caminho e, portanto, tem pelo menos uma aresta em M_B. Logo, $\delta(S) \cup M_B$ é um K-multicorte.

Se $K = \emptyset$ ou $cx = 0$, claramente (3.11) vale. Suponha então que $K \neq \emptyset$ e $cx > 0$. Neste caso, o algoritmo devolve $M := \delta(S) \cup M_B$. Como $\{s, t\}$ está em K mas não em K_B, temos $|K_B| < |K|$. Isso garante o sucesso da recursão na linha 17 e, portanto, (3.11) vale com M_B, c_B, x_B e K_B no lugar de M, c, x e K. Disso, e do lema 3.3, concluímos que

$$
\begin{aligned}
c(\delta(S) \cup M_B) \ = & \\
= \ & c(\delta(S)) + c_B(M_B) \\
\leq \ & (2 \ln 2k)(c_A x_A + c_{\delta(S)} x_{\delta(S)} + \tfrac{1}{k} cx + (1 + \tfrac{1}{k}|K_B|) c_B x_B) \\
= \ & (2 \ln 2k)(c_A x_A + c_{\delta(S)} x_{\delta(S)} + c_B x_B + \tfrac{1}{k} cx + \tfrac{1}{k}|K_B| c_B x_B) \\
= \ & (2 \ln 2k)(cx + \tfrac{1}{k} cx + \tfrac{1}{k}|K_B| c_B x_B) && (3.14) \\
\leq \ & (2 \ln 2k)(cx + \tfrac{1}{k} cx + \tfrac{1}{k}(|K| - 1) c_B x_B) && (3.15) \\
\leq \ & (2 \ln 2k)(1 + \tfrac{1}{k}|K|) cx\,. && (3.16)
\end{aligned}
$$

Como no lema 3.3, $A := E_{G[S]}$. A igualdade (3.14) vale pois $(A, \delta(S), B)$ é uma partição de E_G e portanto $c_A x_A + c_{\delta(S)} x_{\delta(S)} + c_B x_B = cx$. A desigualdade (3.15) vale pois $|K_B| \leq |K| - 1$. Finalmente, a desigualdade (3.16) vale pois $c_B x_B \leq cx$.

Quanto ao tempo de execução do algoritmo CENTRAL, não é difícil ver que as linhas 1 a 16 consomem tempo polinomial em $\langle G \rangle$, $|K|$ e $\langle x \rangle$. Em especial, as linhas 10 e 11 do algoritmo consomem tempo $O(|V_G|)$. Com estas informações, é fácil mostrar, por indução em $|K|$, que o algoritmo consome tempo polinomial em $\langle G \rangle + \langle c \rangle + \langle x \rangle$. □

Exercícios

3.1 Mostre que o algoritmo MINCC-HOCHBAUM não tem uma razão de aproximação menor que o número máximo de vezes, β, que um elemento de E aparece em conjuntos de $\mathcal{S}$. Em outras palavras, exiba uma instância $(E, \mathcal{S}, c)$ do problema MINCC para a qual o algoritmo devolve uma cobertura de custo não inferior a $\beta \operatorname{opt}(E, \mathcal{S}, c)$,

3.2 O **problema da cobertura mínima por vértices** (*minimum vertex cover problem*) consiste no seguinte:

> **Problema** MINCV (G, c): *Dados um grafo G e um custo c_v em $\mathbb{Q}_{\geq}$ para cada vértice v, encontrar um conjunto S de vértices que contenha pelo menos um dos extremos de cada uma das arestas de G e minimize $c(S)$.*

Esse problema pode ser visto como um caso particular do MINCC. Escreva uma versão especializada do MINCC-HOCHBAUM para o MINCV. Qual a menor razão de aproximação do algoritmo?

3.3 Considere a variante do MINCC $(E, \mathcal{S}, c)$, conhecida como **problema da multicobertura mínima por conjuntos** (*minumum set* *multicover problem*), em que cada elemento e de E tem associado a ele um número r_e em $\mathbb{Z}_>$ e a coleção almejada deve conter r_e conjuntos que contenham e. Descreva uma H_n-aproximação, $n := |E|$, para esta variante do MINCC.

3.4 Compare o algoritmo MINCC-HOCHBAUM com o algoritmo MINCC-CHVÁTAL. Qual tem melhor razão de aproximação?

3.5 Aplique a técnica do arredondamento ao problema MOCHILA, introduzido no capítulo 2, demonstrando a melhor razão de aproximação que você puder para o algoritmo obtido.

Notas bibliográficas

A primeira versão deste capítulo baseou-se no livro editado por Hochbaum [Hoc97]. O exercício 3.3 foi tirado do livro de Vazirani [Vaz01].

Srinivasan [Sri99] obteve um outro algoritmo para o MINCC usando um tipo de arredondamento probabilístico (capítulo 6). O problema da multicobertura por conjuntos, apresentado no exercício 3.3, foi considerado e analisado por Rajagopalan e Vazirani [RV99]. Para vários outros problemas [CKR00, Shm98], a técnica de arredondamento tem demonstrado bons resultados.

Leighton e Rao [LR99] introduziram a técnica métrica em seu artigo sobre o problema do multifluxo uniforme(*uniform multicommodity flow problem*). Garg, Vazirani e Yannakakis [GVY96], baseando-se na abordagem feita por Leighton e Rao, obtiveram um algoritmo que é essencialmente o que apresentamos na seção 3.2.

Técnicas métricas são usadas em vários trabalhos [AR98, CKR00, ENRS95, KRAR95, LLR95]. Alguns recorrem, de uma maneira muito sofisticada, a propriedades métricas de certas imersões no $\mathbb{R}^d$, às vezes combinadas com técnicas probabilísticas. O leitor interessado pode consultar a resenha de Shmoys [Hoc97], onde resultados nessa linha são apresentados.

O algoritmo que resolve o MINMCUT (G, K, c) quando $|K| = 1$ baseia-se no teorema do fluxo máximo e corte mínimo [AMO93, FF56]. Uma variante desse teorema vale no caso em que $|K| = 2$ [Hu63, Ita78] e, como conseqüência, há um algoritmo polinomial também para este caso.

O MINMCUT é NP-difícil também quando restrito a árvores binárias com custos unitários [CFR98, DJP$^+$94]. Para árvores, há uma 2-aproximação polinomial [GVY97] (exercício 5.15). Uma α-aproximação polinomial com α constante e menor que 2 para o caso das árvores resolveria um problema bem-conhecido e em aberto há anos: o de encontrar um algoritmo de aproximação com razão menor que 2 para o problema MINCV da cobertura mínima por vértices (exercício 8.3).

Uma variante muito interessante do MINMCUT é conhecida como *multiterminal-cut* ou *multiway-cut*: dados um grafo G, um conjunto L de vértices e um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e, encontrar um conjunto de arestas de custo mínimo cuja remoção deixa cada vértice de L em um componente diferente. Dahlhaus *et al.* [DJP+94] mostraram que o problema é NP-difícil mesmo quando $|L| = 3$. (Esta variante pode ser reduzida ao MINMCUT se substituirmos o conjunto L pelo conjunto de todos os pares de vértices em L. Disso segue que MINMCUT (G, K, c) é NP-difícil quando $|K| = 3$.) Eles projetaram também uma $(2 - 2/|L|)$-aproximação polinomial para esse problema. Atualmente a melhor razão de aproximação conhecida para esse problema é 1,3438 [KKS+99].

Capítulo 4

Método Dual

Neste capítulo descrevemos como obter algoritmos de aproximação para um problema de otimização usando o dual de um programa linear que representa o problema em questão. Esta estratégia de projeto de algoritmos de aproximação é chamada de método dual.

4.1 Cobertura por vértices

O problema que vamos usar para apresentar o método é o problema da cobertura mínima por vértices. Dado um grafo G, uma **cobertura por vértices** é um conjunto de vértices com pelo menos um dos extremos de cada aresta. O **problema da cobertura mínima por vértices** (*vertex cover problem*) consiste no seguinte:

> **Problema** MinCV (G, c): *Dados um grafo G e um custo c_v em $\mathbb{Q}_{\geq}$ para cada vértice v, encontrar uma cobertura por vértices S que minimize $c(S)$.*

O problema é NP-difícil mesmo quando o grafo é conexo, planar e tem grau máximo 4 [GJ79]. Para este e outros problemas semelhantes, o método dual se aplica de forma simples e direta. A sua simplicidade não o impede de ser eficaz: o método dual produz o melhor algoritmo de aproximação conhecido para o MinCV.

O método consiste em obter uma solução ótima do dual de uma relaxação linear do problema original e, a partir desta solução, produzir uma resposta para o problema original. Considere, pois, a seguinte relaxação linear do MINCV: encontrar um vetor x indexado por V_G que

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & x_u + x_v & \geq & 1 & \text{para cada } uv \text{ em } E_G\,, \\ & x_v & \geq & 0 & \text{para cada } v \text{ em } V_G\,. \end{array} \tag{4.1}$$

O dual deste programa é encontrar um vetor y indexado por E_G que

$$\begin{array}{rrcll} \text{maximize} & y(E_G) & & & \\ \text{sob as restrições} & y(\delta(v)) & \leq & c_v & \text{para cada } v \text{ em } V_G\,, \\ & y_e & \geq & 0 & \text{para cada } e \text{ em } E_G\,. \end{array} \tag{4.2}$$

O programa (4.1) é viável pois o vetor característico de V_G satisfaz as restrições. O programa (4.2) é viável pois o vetor nulo satisfaz as restrições. Assim, pelo teorema forte da dualidade (teorema C.3, apêndice C), o programa (4.2) tem solução ótima racional. Além disso, o vetor característico de qualquer cobertura por vértices é viável em (4.1) e, portanto, temos que

$$\mathrm{opt}(G,c) \geq c\hat{x} = \hat{y}(E_G)\,, \tag{4.3}$$

para qualquer solução ótima $\hat{x}$ de (4.1) e qualquer solução ótima $\hat{y}$ de (4.2).

Uma vez obtida uma solução ótima $\hat{y}$ de (4.2), escolha todos os vértices de G cuja restrição em (4.2) é satisfeita com igualdade por $\hat{y}$. É fácil justificar essa estratégia a partir das condições de folgas complementares (condição C.5) para (4.1) e (4.2): se $\hat{x}$ é uma solução ótima do programa (4.1) então todo vértice v tal que $\hat{x}_v > 0$ é escolhido. O algoritmo resultante foi proposto por Hochbaum [Hoc82] originalmente para o MINCC, que generaliza o MINCV.

Algoritmo MINCV-HOCHBAUM (G, c)

1 seja $\hat{y}$ uma solução ótima racional de (4.2)
2 $C \leftarrow \{v \in V_G : \hat{y}(\delta(v)) = c_v\}$
3 devolva C

A prova do teorema abaixo é semelhante à prova do lema das folgas complementares (lema C.2).

Teorema 4.1: *O algoritmo* MINCV-HOCHBAUM *produz uma cobertura por vértices.*

Demonstração: Seja C o conjunto devolvido pelo algoritmo. Denote por d_v a folga na restrição em (4.2) que corresponde ao vértice v, ou seja,

$d_v := c_v - \hat{y}(\delta(v))$. Considere uma aresta arbitrária $f = ij$ e seja $\varepsilon := \min\{d_i, d_j\}$. Queremos mostrar que $\varepsilon = 0$. Defina $\dot{y}_f := \hat{y}_f + \varepsilon$ e $\dot{y}_e := \hat{y}_e$ para toda aresta e distinta de f. Note que $\dot{y}(\delta(v)) = \hat{y}(\delta(v)) \leq c_v$, para todo vértice v distinto de i e j. Além disso, para v em $\{i, j\}$, temos que $\dot{y}(\delta(v)) = \hat{y}(\delta(v)) + \varepsilon \leq c_v$. Isso significa que $\dot{y}$ é viável em (4.2). Como $\hat{y}$ é uma solução ótima de (4.2) e $\dot{y}(E_G) = \hat{y}(E_G) + \varepsilon$, temos que $\varepsilon = 0$. Portanto, uma das restrições em (4.2) que correspondem aos vértices i e j não tem folga. Assim, C é uma cobertura por vértices. □

A qualidade da solução produzida por este algoritmo é atestada pelo seguinte teorema.

Teorema 4.2: *O algoritmo* MinCV-Hochbaum *é uma* 2*-aproximação polinomial para o* MinCV.

Demonstração: Ao final do algoritmo, C é tal que

$$c(C) \;=\; \textstyle\sum_{v \in C} c_v \;=\; \sum_{v \in C} \hat{y}(\delta(v)) \;\leq\; 2\,\hat{y}(E_G) \;\leq\; 2\,\mathrm{opt}(G, c)\,.$$

A primeira desigualdade vale pois, para cada aresta $e = ij$, a variável $\hat{y}_e$ aparece no máximo duas vezes na soma: uma vez se i está em C e outra se j está em C. A segunda desigualdade vale por (4.3).

O programa linear (4.2) tem $|E_G|$ variáveis e $|V_G|$ restrições. Assim, a linha 1 do algoritmo pode ser executada em tempo polinomial em $\langle G \rangle + \langle c \rangle$ (fato C.4). Claramente os demais passos do algoritmo podem ser executados em tempo polinomial em $\langle G \rangle + \langle c \rangle$. Podemos então concluir que o MinCV-Hochbaum é polinomial. □

A melhor razão de aproximação conhecida para o MinCV é 2, ou seja, não se conhece um algoritmo para o MinCV melhor, em termos de razão de aproximação, que o MinCV-Hochbaum.

Exercícios

4.1 Mostre que o algoritmo MinCV-Hochbaum não tem uma razão de aproximação menor que 2, ou seja, apresente uma instância (G, c) do MinCV para a qual o algoritmo devolva uma cobertura por vértices S em que $c(S) = 2\,\mathrm{opt}(G, c)$.

4.2 Mostre que o algoritmo MinCV-Hochbaum é diferente do algoritmo sugerido no exercício 3.2, ou seja, exiba uma instância do problema onde os algoritmos produzem (ou podem produzir) coberturas diferentes.

4.3 Aplique o método dual ao problema da cobertura mínima por conjuntos (MINCC), definido na seção 2.2. Mostre que o algoritmo resultante é uma β-aproximação, onde β é o número máximo de conjuntos em que um elemento aparece.

4.4 Considere o **problema da cobertura mínima por arestas** (*minimum edge cover problem*):

> **Problema** MINCA (G, c): *Dados um grafo G e um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e, encontrar um conjunto S de arestas tal que todo vértice pertence a alguma aresta de S e $c(S)$ é mínimo.*

O MINCA é um caso particular "fácil" do MINCC: existe um algoritmo polinomial que o resolve [Law76]. Mostre que o método dual dá uma Δ-aproximação polinomial para o MINCA, onde Δ é o grau máximo em G.

4.5 Dê uma prova alternativa do teorema 2.2 que use o dual do programa (3.1). Mais precisamente, encontre uma solução viável do dual do programa (3.1) cujo valor é igual ao da cobertura produzida pelo algoritmo MINCC-CHVÁTAL. Disso e do teorema fraco da dualidade (lema C.1), deduza o teorema 2.2.

Notas bibliográficas

Como já mencionamos, o problema da cobertura mínima por vértices (MINCV) é um caso especial do problema da cobertura mínima por conjuntos (MINCC). Este último foi extensivamente investigado por Hochbaum [Hoc82] que apresentou, como vimos no capítulo 3, uma β-aproximação polinomial para o problema. Aqui, β é o número máximo de vezes que um elemento de E aparece em conjuntos de $\mathcal{S}$, quando se considera uma instância $(E, \mathcal{S}, c)$.

Além da abordagem primal, vista no capítulo 3, Hochbaum apresentou uma β-aproximação para o MINCC baseado na formulação dual. Esse algoritmo é essencialmente uma generalização do que discutimos para o MINCV. Mais recentemente, Hochbaum [Hoc97] escreveu uma resenha completa sobre algoritmos produzidos pelos métodos primal e dual para vários problemas de cobertura. Hall e Hochbaum [HH86] obtiveram algoritmos, baseados tanto no método primal quanto no dual, para o problema da multicobertura mínima por conjuntos (exercício 3.3).

A prova alternativa da razão de aproximação do MinCC-Chvátal sugerida no exercício 4.5 deve-se a Chvátal [Chv79] e Lovász [Lov75].

O esquema de aproximação polinomial de Hochbaum e Shmoys [HS87, HS88] para o Escalonamento, mencionado nas notas do capítulo 2, usa uma abordagem que eles chamaram de *dual approximation*. Essa abordagem difere do método dual aqui descrito. Ela consiste em encontrar soluções duais super-ótimas (inviáveis, portanto) que são então convertidas em boas soluções viáveis.

Capítulo 5

Método Primal-Dual

Este capítulo apresenta o método primal-dual clássico e uma generalização dele que tem sido usada com bastante sucesso no projeto de algoritmos de aproximação: o método de aproximação primal-dual.

O método primal-dual reduz, apoiando-se nas condições de folgas complementares, um problema de programação linear a uma seqüência de problemas de viabilidade, potencialmente mais simples. Nas aplicações do método em otimização combinatória, esses problemas de viabilidade são muitas vezes outros problemas de otimização combinatória para os quais são conhecidos algoritmos eficientes, não raro *puramente combinatórios*. Por algoritmo puramente combinatório entenda-se um algoritmo sem uma referência explícita a algum algoritmo de programação linear, como os que vimos nos capítulos 3 e 4.

Neste capítulo, mostramos os algoritmo obtidos da aplicação do método de aproximação primal-dual ao problema da transversal mínima e ao problema da floresta de Steiner.

5.1 Método primal-dual clássico

Sejam M e N os conjuntos de índices de linhas e colunas de uma matriz A. Sejam ainda b um vetor indexado por M e c um vetor indexado por N. Considere o seguinte problema de programação linear, que será denotado por $\mathrm{P}(A,b,c)$: encontrar um vetor x indexado por N que $\mathrm{P}(A,b,c)$

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & (Ax)_i & \geq & b_i & \text{para cada } i \text{ em } M\,, \\ & x_j & \geq & 0 & \text{para cada } j \text{ em } N\,. \end{array} \qquad (5.1)$$

O conjunto das soluções viáveis do problema $\mathrm{P}(A,b,c)$ será denotado por
$X(A,b)$ $X(A,b)$. O problema $\mathrm{P}(A,b,c)$ é viável se e somente se $X(A,b)$ não é vazio. O dual do problema $\mathrm{P}(A,b,c)$ consiste em encontrar um vetor y indexado por M que

$$\begin{array}{rrcll} \text{maximize} & y\,b & & & \\ \text{sob as restrições} & (yA)_j & \le & c_j & \text{para cada } j \text{ em } N\,, \\ & y_i & \ge & 0 & \text{para cada } i \text{ em } M\,. \end{array} \tag{5.2}$$

$\mathrm{D}(A,c,b)$ $Y(A,c)$ Será usada a abreviatura $\mathrm{D}(A,c,b)$ para denotar esse programa linear e $Y(A,c)$ para representar o conjunto de suas soluções viáveis. O problema $\mathrm{D}(A,c,b)$ é viável se e somente se $Y(A,c)$ não é vazio. Dizemos que $\mathrm{D}(A,c,b)$ é o programa **dual** e $\mathrm{P}(A,b,c)$ é o programa **primal**.

Dois vetores x e y, indexados por M e N respectivamente, têm **folgas complementares** se, para cada índice j em N, tem-se que

$$x_j = 0 \quad \text{ou} \quad (yA)_j = c_j$$

e, para cada índice i em M, tem-se que

$$y_i = 0 \quad \text{ou} \quad (Ax)_i = b_i\,.$$

O lema das folgas complementares (lema C.2, apêndice C) afirma que, para todo x em $X(A,b)$ e todo y em $Y(A,c)$, tem-se $cx = y\,b$ se e somente se as folgas de x e y são complementares.

A idéia geral do método primal-dual é a seguinte. O método é iterativo e no início de cada iteração tem-se um vetor y viável no programa dual. Cada iteração consiste na procura por um vetor x viável no programa primal que tem folgas complementares às de y. Se um tal vetor x é encontrado, o método pára, pois x e y são soluções ótimas dos programas primal e dual, respectivamente. Se um tal vetor x não é encontrado, o método modifica y para obter um novo vetor z viável no programa dual e começa uma nova iteração com z no papel de y.

O método primal-dual recebe um sistema (A,b,c) tal que $Y(A,c)$ não é vazio e devolve: (1) vetores x em $X(A,b)$ e y em $Y(A,c)$ tais que $cx = y\,b$, ou (2) um vetor y' em $Y(A,0)$ tal que $y'\,b > 0$.

Cada iteração do método começa com um vetor y em $Y(A,c)$. No início da primeira iteração, y é um elemento qualquer de $Y(A,c)$. Cada iteração consiste no seguinte. Sejam

$I(y)$ $J(y)$

$$I(y) := \{i \in M : y_i = 0\} \quad \text{e} \quad J(y) := \{j \in N : (yA)_j = c_j\}\,.$$

Considere o **problema restrito primal** a seguir, denotado por $\mathrm{RP}(A,b,y)$: encontrar um vetor x indexado por N tal que RP( )

$$
\begin{aligned}
(Ax)_i &\geq b_i \quad \text{para cada } i \text{ em } I(y)\,,\\
(Ax)_i &= b_i \quad \text{para cada } i \text{ em } M \setminus I(y)\,,\\
x_j &\geq 0 \quad \text{para cada } j \text{ em } J(y)\,,\\
x_j &= 0 \quad \text{para cada } j \text{ em } N \setminus J(y)\,.
\end{aligned}
\tag{5.3}
$$

Este é um problema de viabilidade: trata-se de encontrar x que satisfaz as restrições (5.3). Suponha que o problema $\mathrm{RP}(A,b,y)$ é viável e seja x uma de suas soluções. Um tal vetor x está em $X(A,b)$ e tem folgas complementares às de y, donde $cx = yb$. Nesta situação o método pára após devolver os vetores x e y que, de acordo com o teorema fraco da dualidade (lema C.1, apêndice C), são soluções ótimas do problema $\mathrm{P}(A,b,c)$ e $\mathrm{D}(A,c,b)$, respectivamente.

Se o problema $\mathrm{RP}(A,b,y)$ é inviável então, pelo lema de Farkas (lema C.5, apêndice C), o seguinte problema de viabilidade tem solução: encontrar um vetor y' indexado por M tal que

$$
\begin{aligned}
y'b &> 0\,,\\
(y'A)_j &\leq 0 \quad \text{para cada } j \text{ em } J(y)\,,\\
y'_i &\geq 0 \quad \text{para cada } i \text{ em } I(y)\,.
\end{aligned}
\tag{5.4}
$$

Esse problema é chamado de **problema restrito dual** e será denotado por $\mathrm{RD}(A,b,y)$. RD( )

Seja y' uma solução do problema $\mathrm{RD}(A,b,y)$. Se, para todo número positivo θ, o vetor $y+\theta y'$ está em $Y(A,c)$, então o programa linear $\mathrm{D}(A,c,b)$ é ilimitado e o método pára após devolver y', vetor de inviabilidade para $\mathrm{P}(A,b,c)$. Caso contrário, o método começa uma nova iteração com $y+\theta y'$ no papel de y, onde θ é o maior número tal que $y+\theta y'$ está em $Y(A,c)$. Abaixo encontra-se a descrição do método.

Método Primal-Dual (A,b,c)

```
seja y um vetor em Y(A,c)
enquanto RP(A,b,y) não tem solução faça
    seja y' uma solução do RD(A,b,y)
    se y + θy' ∈ Y(A,c) para todo θ positivo
        então devolva y'
        senão seja θ o maior número tal que y + θy' ∈ Y(A,c)
              y ← y + θy'
seja x uma solução do RP(A,b,y)
devolva x e y
```

5.2 Método de aproximação primal-dual

O método de aproximação primal-dual é semelhante ao método clássico, com a diferença que o método pára assim que encontra soluções viáveis dos problemas $\mathrm{P}(A,b,c)$ e $\mathrm{D}(A,c,b)$ suficientemente próximas das soluções ótimas. Esta proximidade será medida através de folgas aproximadas.

Sejam α e β dois números tais que $0 < \alpha \leq 1 \leq \beta$. Dizemos que x e y têm **folgas α-aproximadas** no primal se, para cada índice j em N, tem-se que

$$x_j = 0 \quad \text{ou} \quad (yA)_j \geq \alpha c_j \,.$$

Dizemos que x e y têm **folgas β-aproximadas** no dual se, para cada índice i em M, tem-se que

$$y_i = 0 \quad \text{ou} \quad (Ax)_i \leq \beta b_i \,.$$

O lema a seguir é uma generalização do lema das folgas complementares.

> **Lema 5.1** (das folgas aproximadas): *Se os vetores não-negativos x e y satisfazem as condições de folgas α-aproximadas no primal e β-aproximadas no dual, então $\alpha\, cx \leq \beta\, y b$.*

Demonstração: Em virtude das folgas aproximadas,

$$\alpha\, cx \;\leq\; (yA)x \;=\; y(Ax) \;\leq\; \beta\, y b\,,$$

onde a primeira desigualdade segue da não-negatividade de x e a segunda da não-negatividade de y. □

Uma conseqüência imediata do lema das folgas aproximadas é o seguinte. Para todo x em $X(A,b)$ e y em $Y(A,c)$ satisfazendo as condições de folgas α-aproximadas no primal e β-aproximadas no dual, tem-se que

$$cx \leq (\beta/\alpha)\, c\hat{x} \quad \text{e} \quad yb \geq (\alpha/\beta)\, \hat{y} b\,,$$

onde $\hat{x}$ e $\hat{y}$ são soluções ótimas de $\mathrm{P}(A,b,c)$ e $\mathrm{D}(A,c,b)$, respectivamente.

O método de aproximação primal-dual recebe um sistema (A,b,c) tal que $Y(A,c)$ não é vazio e dois números α e β tais que $0 < \alpha \leq 1 \leq \beta$ e devolve: (1) vetores x em $X(A,b)$ e y em $Y(A,c)$ tais que $\alpha\, cx \leq \beta\, yb$ ou (2) um vetor y' em $Y(A,0)$ tal que $y'b > 0$.

Cada iteração do método começa com um vetor y em $Y(A,c)$. No início da primeira iteração, y é um elemento qualquer de $Y(A,c)$. Cada iteração consiste no seguinte. Sejam

$I(y)$

$$I(y) := \{i \in M : y_i = 0\} \quad \text{e} \quad J(y,\alpha) := \{j \in N : (yA)_j \geq \alpha c_j\ \}\ .$$ $J(y,\alpha)$

Considere o seguinte **problema restrito aproximado primal**: encontrar um vetor x indexado por N tal que

$$\begin{aligned} (Ax)_i &\geq b_i && \text{para cada } i \text{ em } M\ , \\ (Ax)_i &\leq \beta b_i && \text{para cada } i \text{ em } M \setminus I(y)\ , \\ x_j &\geq 0 && \text{para cada } j \text{ em } J(y,\alpha)\ , \\ x_j &= 0 && \text{para cada } j \text{ em } N \setminus J(y,\alpha)\ . \end{aligned}$$

Será usada a abreviatura $\text{RAP}(A,b,y,\alpha,\beta)$ para denotar esse problema. RAP()

Se o problema $\text{RAP}(A,b,y,\alpha,\beta)$ é viável, então, pelo lema das folgas aproximadas, x é um vetor em $X(A,b)$ tal que $\alpha\, cx \leq \beta\, y b$ e o método devolve x e y. Caso contrário, pelo lema de Farkas, o seguinte **problema restrito aproximado dual**, denotado por $\text{RAD}(A,b,y,\alpha)$, é viável: RAD()
encontrar um vetor y' indexado por M tal que

$$\begin{aligned} y'b &> 0\ , \\ (y'A)_j &\leq 0 && \text{para cada } j \text{ em } J(y,\alpha)\ , \\ y'_i &\geq 0 && \text{para cada } i \text{ em } I(y)\ . \end{aligned}$$

Seja y' uma solução do problema $\text{RAD}(A,b,y,\alpha)$. Se para todo número positivo θ tem-se que $y + \theta y'$ está em $Y(A,c)$, então o programa linear $\text{D}(A,c,b)$ é ilimitado e o método pára após devolver y', vetor de inviabilidade para $\text{P}(A,b,c)$. Caso contrário, seja θ o maior número tal que $y + \theta y'$ está em $Y(A,c)$. O método começa um nova iteração com $y + \theta y'$ no papel de y. Abaixo encontra-se a descrição do método.

Método APROXIMAÇÃO-PRIMAL-DUAL (A,b,c,α,β)

```
seja y um vetor em Y(A,c)
enquanto RAP(A,b,y,α,β) não tem solução faça
    seja y' uma solução do RAD(A,b,y,α)
    se y + θy' ∈ Y(A,c) para todo θ positivo
        então devolva y'
        senão seja θ o maior número tal que y + θy' ∈ Y(A,c)
                y ← y + θy'
seja x uma solução do RAP(A,b,y,α,β)
devolva x e y
```

Suponha que o método pare após devolver os vetores x e y. É fácil verificar que x está em $X(A,b)$, que y está em $Y(A,c)$ e que os vetores x e

y têm folgas α-aproximadas no primal e β-aproximadas no dual. Donde, pelo lema das folgas aproximadas, tem-se $\alpha\, cx \le \beta\, y\, b$. Assim, o vetor y é um certificado da qualidade de x como solução viável de $\mathrm{P}(A, b, c)$, já que

$$cx \;\le\; (\beta/\alpha)\, y\, b \;\le\; (\beta/\alpha)\, \hat{y}\, b \;=\; (\beta/\alpha)\, c\hat{x}\,,$$

onde $\hat{x}$ e $\hat{y}$ são soluções ótimas de $\mathrm{P}(A, b, c)$ e $\mathrm{D}(A, c, b)$, respectivamente. Da mesma forma, o vetor x certifica que $y\, b \ge (\alpha/\beta)\, \hat{y}\, b$.

5.3 Transversal mínima

Seja $\mathcal{S}$ uma coleção finita de subconjuntos de um conjunto finito E. Um subconjunto T de E é uma **transversal** de $\mathcal{S}$ se $T \cap S$ é não-vazio para cada S em $\mathcal{S}$. O **problema da transversal mínima** (*hitting set problem*) consiste no seguinte:

> **Problema** MINTC $(E, \mathcal{S}, c)$: *Dados um conjunto finito E, uma coleção finita $\mathcal{S}$ de subconjuntos de E e um custo c_e em $\mathbb{Q}_{\ge}$ para cada e em E, encontrar uma transversal T de $\mathcal{S}$ que minimize $c(T)$.*

Diversos problemas combinatórios são casos particulares do MINTC; veja os exercícios no fim do capítulo. Às vezes, dizemos que $c(T)$ é o **custo** da transversal T. Com isso, o problema consiste em encontrar uma transversal de $\mathcal{S}$ de custo mínimo.

O seguinte programa linear é uma relaxação de MINTC $(E, \mathcal{S}, c)$: encontrar um vetor x indexado por E que

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & x(S) & \ge & 1 & \text{para cada } S \text{ em } \mathcal{S}\,, \\ & x_e & \ge & 0 & \text{para cada } e \text{ em } E\,. \end{array} \tag{5.5}$$

De fato, o vetor característico x de qualquer transversal T é um vetor viável de (5.5) de custo $c(T)$. O correspondente programa dual consiste em encontrar um vetor y indexado por $\mathcal{S}$ que

$$\begin{array}{rrcll} \text{maximize} & y(\mathcal{S}) & & & \\ \text{sob as restrições} & \sum_{S\,:\,e \in S} y_S & \le & c_e & \text{para cada } e \text{ em } E\,, \\ & y_S & \ge & 0 & \text{para cada } S \text{ em } \mathcal{S}\,. \end{array} \tag{5.6}$$

β Tome $\beta := \max_{S \in \mathcal{S}} |S|$ e seja y uma solução viável do problema (5.6).

Sejam ainda

$$I(y) := \{S \in \mathcal{S} : y_S = 0\} \quad \text{e} \quad J(y) := \{e \in E : \textstyle\sum_{S\,:\,e\in S} y_S \geq c_e\}\ .$$

O problema restrito aproximado relativo a y, associado ao problema (5.5), consiste em encontrar um vetor x indexado por E tal que

$$\begin{aligned} x(S) &\geq 1 && \text{para cada } S \text{ em } \mathcal{S}\ , \\ x(S) &\leq \beta && \text{para cada } S \text{ em } \mathcal{S} \setminus I(y)\ , \\ x_e &\geq 0 && \text{para cada } e \text{ em } J(y)\ , \\ x_e &= 0 && \text{para cada } e \text{ em } E \setminus J(y)\ . \end{aligned}$$

Podemos trocar S por $J(y) \cap S$ nas duas primeiras desigualdades por causa da última igualdade. Assim, esse problema pode ser reescrito como

$$\begin{aligned} x(J(y) \cap S) &\geq 1 && \text{para cada } S \text{ em } \mathcal{S}\ , \\ x(J(y) \cap S) &\leq \beta && \text{para cada } S \text{ em } \mathcal{S} \setminus I(y)\ , \\ x_e &\geq 0 && \text{para cada } e \text{ em } J(y)\ , \\ x_e &= 0 && \text{para cada } e \text{ em } E \setminus J(y)\ . \end{aligned} \tag{5.7}$$

Se o vetor característico x de $J(y)$ é viável em (5.7), então $J(y)$ é uma transversal de $\mathcal{S}$. Se $J(y)$ não é uma transversal é evidente que o sistema (5.7) é inviável. Esta observação é suficiente para obter uma solução viável do seguinte problema restrito aproximado dual: encontrar um vetor y' indexado por $\mathcal{S}$ tal que

$$\begin{aligned} y'(\mathcal{S}) &> 0 && , \\ \textstyle\sum_{S\,:\,e\in S} y'_S &\leq 0 && \text{para cada } e \text{ em } J(y)\ , \\ y'_S &\geq 0 && \text{para cada } S \text{ em } I(y)\ . \end{aligned} \tag{5.8}$$

Se $J(y) \cap R$ é vazio para algum R, então o vetor característico y' de $\{R\}$ é uma solução de (5.8).

O algoritmo resultante da aplicação do método de aproximação primal-dual ao MinTC é devido a Bar-Yehuda e Even [BYE81] e foi originalmente concebido para o problema da cobertura mínima por vértices. O algoritmo MinTC-BE recebe um conjunto finito E, uma coleção $\mathcal{S}$ de subconjuntos não-vazios de E e um custo c_e em $\mathbb{Q}_{\geq}$ para cada e em E e devolve uma transversal J de $\mathcal{S}$ tal que $c(J) \leq \beta\, \mathrm{opt}(E, \mathcal{S}, c)$. A descrição do algoritmo supõe que cada subconjunto de $\mathcal{S}$ não é vazio, e portanto o problema MinTC$(E, \mathcal{S}, c)$ é viável.

Algoritmo MINTC-BE $(E, \mathcal{S}, c)$

1 $J \leftarrow \{e \in E : c_e = 0\}$
2 para cada S em $\mathcal{S}$ faça $y_S \leftarrow 0$
3 enquanto existe R em $\mathcal{S}$ tal que $J \cap R = \emptyset$ faça
4   seja y' o vetor característico de $\{R\}$
5   seja θ o maior número tal que
6         $\sum_{S\,:\,e \in S} (y + \theta y')_S \leq c_e$ para cada elemento e de R
7   seja f um elemento de R tal que $\sum_{S\,:\,f \in S} (y + \theta y')_S = c_f$
8     $y \leftarrow y + \theta y'$
9     $J \leftarrow J \cup \{f\}$
10 devolva J

Teorema 5.2: *O algoritmo* MINTC-BE *é uma β-aproximação polinomial para o* MINTC $(E, \mathcal{S}, c)$, *onde* $\beta := \max_{S \in \mathcal{S}} |S|$.

Demonstração: Durante a execução do algoritmo, y é uma solução viável do programa linear (5.6). É evidente que o conjunto J devolvido pelo algoritmo é uma transversal de $\mathcal{S}$. Ademais, pela escolha de β, é claro que $|J \cap S| \leq \beta$ para cada S em $\mathcal{S}$. Logo, se x é o vetor característico de J e $\hat{x}$ é uma solução ótima de (5.5), então

$$c(J) \;=\; cx \;\leq\; \beta\, y\, b \;\leq\; \beta\, c\hat{x} \;\leq\; \beta \operatorname{opt}(E, \mathcal{S}, c)\;,$$

onde a primeira desigualdade segue do lema das folgas aproximadas e a segunda do teorema fraco da dualidade.

Temos que $|J|$ cresce a cada execução da linha 9, donde o número de execuções das linhas 3 a 9 do algoritmo é no máximo $|E|$. Ademais, a execução de cada linha consome tempo $O(|E||\mathcal{S}|)$. $\square$

A transversal J devolvida pelo algoritmo pode não ser minimal. É concebível que exista uma transversal propriamente contida em J de custo menor que $c(J)$. Um pós-processamento simples pode extrair de J uma transversal minimal. Um tal pós-processamento é louvável do ponto de vista prático, mas não resulta em uma melhor razão de aproximação para o algoritmo MINTC-BE.

A próxima seção trata de um problema que é um caso particular do MINTC. O método de aproximação primal-dual aplicado a este problema particular resultará em um algoritmo que, embora semelhante em muitos aspectos ao algoritmo desta seção, tem uma razão de aproximação substancialmente melhor. Esta melhora se deve em parte à liberdade que o método deixa para que estruturas especiais sejam exploradas para, por exemplo, determinar os parâmetros α e β e escolher o vetor y'. Curiosamente, como será visto, o mesmo pós-processamento que é inoperante para o algoritmo MINTC-BE é crucial para o algoritmo da próxima seção; este fenômeno é compreensível tendo em vista a maior estrutura da coleção $\mathcal{S}$.

5.4 Floresta de Steiner

Seja G um grafo e $\mathcal{R}$ uma coleção de subconjuntos de V_G. Uma $\mathcal{R}$-**floresta** de G é qualquer floresta geradora[1] F de G tal que todo elemento de $\mathcal{R}$ está contido em algum componente de F. Quando $\mathcal{R}$ está subentendida, dizemos simplesmente que F é uma **floresta de Steiner** de G. Se $\mathcal{R}$ tem um só elemento, o conceito de floresta de Steiner reduz-se ao de **árvore de Steiner**: uma árvore (não necessariamente geradora) de G cujo conjunto de vértices contém o único conjunto em $\mathcal{R}$.

O **problema da floresta de Steiner** (*Steiner forest problem*) consiste no seguinte:

> **Problema** MINFS $(G, c, \mathcal{R})$: *Dados um grafo G, um custo c_e em $\mathbb{Q}_\geq$ para cada aresta e e uma coleção $\mathcal{R}$ de subconjuntos de V_G, encontrar uma $\mathcal{R}$-floresta F que minimize $c(F)$.*

Podemos dizer que $c(F)$ é o **custo** da floresta F. Assim, o problema consiste em encontrar uma floresta de Steiner de custo mínimo.

O problema MINFS é NP-difícil, mesmo quando restrito a árvores de Steiner, ou seja, mesmo quando a coleção $\mathcal{R}$ contém um só conjunto [GJ79].

Dada uma instância $(G, c, \mathcal{R})$ do problema, adotamos as abreviaturas $V := V_G$ e $E := E_G$ e dizemos que um subconjunto S de V é **ativo** se

$$R \cap S \neq \emptyset \quad \text{e} \quad R \setminus S \neq \emptyset\,.$$

para algum R em $\mathcal{R}$. A coleção de todos os conjuntos ativos será denotada por $\mathcal{S}$. É fácil verificar que uma floresta geradora F de G é uma $\mathcal{R}$-floresta $\mathcal{S}$
se e somente se

$$\delta_F(S) \neq \emptyset \quad \text{para todo} \quad S \quad \text{em} \quad \mathcal{S}\,. \tag{5.9}$$

[1] A rigor, não há necessidade de exigir que a floresta seja geradora; mas essa condição é conveniente.

Portanto, toda floresta de Steiner é uma transversal da coleção de todos os cortes da forma $\delta(S)$ com S em $\mathcal{S}$. Um subconjunto de V que não pertence a $\mathcal{S}$ é **inativo**. É claro que uma floresta geradora F é uma $\mathcal{R}$-floresta se e somente se V_T é inativo para cada componente T de F.

O seguinte programa linear é uma relaxação de MINFS $(G, c, \mathcal{R})$: encontrar um vetor x indexado por E que

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & x(\delta(S)) & \geq & 1 & \text{para cada } S \text{ em } \mathcal{S}\,, \\ & x_e & \geq & 0 & \text{para cada } e \text{ em } E\,. \end{array} \tag{5.10}$$

Dada uma floresta de Steiner F, é evidente que o vetor característico de E_F é uma solução viável de (5.10). Portanto, se $\hat{x}$ é uma solução ótima de (5.10) então $c\hat{x} \leq \text{opt}(G, c, \mathcal{R})$.

O dual do programa (5.10) consiste em encontrar um vetor y indexado pela coleção $\mathcal{S}$ dos subconjuntos ativos de V que

$$\begin{array}{rrcll} \text{maximize} & y(\mathcal{S}) & & & \\ \text{sob as restrições} & \sum_{S\,:\,e\in\delta(S)} y_S & \leq & c_e & \text{para cada } e \text{ em } E\,, \\ & y_S & \geq & 0 & \text{para cada } S \text{ em } \mathcal{S}\,. \end{array} \tag{5.11}$$

Se y é uma solução viável de (5.11) e $\hat{x}$ é uma solução ótima de (5.10) então $y(\mathcal{S}) \leq c\hat{x}$, de acordo com o teorema fraco da dualidade. Portanto,

$$y(\mathcal{S}) \;\leq\; \text{opt}(G, c, \mathcal{R})\,. \tag{5.12}$$

Essa delimitação inferior de $\text{opt}(G, c, \mathcal{R})$ é fundamental para o cálculo da razão de aproximação do algoritmo que discutimos mais abaixo, devido a Goemans e Williamson [GW95a].

Considere agora as condições de folgas aproximadas. A condição de folgas 1-aproximadas no primal exige que

$$\textstyle\sum_{S\,:\,e\in\delta(S)} y_S = c_e \;\text{ sempre que }\; x_e > 0 \tag{5.13}$$

e a condição de folgas 2-aproximadas no dual exige que

$$x(\delta(S)) \leq 2 \;\text{ sempre que }\; y_S > 0\,. \tag{5.14}$$

Se soubéssemos encontrar uma solução viável x de (5.10) e uma solução viável y de (5.11) que satisfizessem as duas condições de folgas aproximadas, teríamos uma 2-aproximação para o MINFS. Sabemos como satisfazer a condição (5.13), mas não sabemos como satisfazer (5.14). Ainda assim, é possível obter uma 2-aproximação se respeitarmos a condição (5.14) em um certo sentido "médio".

Descrição do algoritmo

No início de cada iteração, temos uma floresta geradora F de G e um vetor y indexado por $\mathcal{S}$. O vetor y é viável em (5.11) e o par (F, y) satisfaz a condição de folgas 1-aproximadas

$$\textstyle\sum_{S\,:\,e\in\delta(S)} y_S = c_e \text{ para cada } e \text{ em } F\,, \tag{5.15}$$

que corresponde a (5.13). Em cada iteração, dizemos que um componente T de F é **ativo** se $V_T \in \mathcal{S}$, e **inativo** em caso contrário. Denotamos por $\mathcal{S}_F$ a coleção dos conjuntos de vértices dos componentes ativos de F. $\mathcal{S}_F$ Abusando um pouco da terminologia, dizemos que $\mathcal{S}_F$ é a coleção dos componentes ativos de F. Cada elemento S de $\mathcal{S}_F$ viola a restrição $x(\delta(S)) \geq 1$ de (5.10), onde x é o vetor característico de F; isso sugere que devemos acrescentar à F alguma das arestas de $\delta(S)$. Qualquer aresta desse tipo liga dois componentes distintos de F. Dizemos que uma tal aresta é **externa** à F. Devemos, então, escolher uma aresta externa e acrescentá-la à F. Resta esclarecer como escolher essa aresta.

Como o programa dual (5.11) tem o objetivo de maximizar a soma das variáveis y_S, procuramos aumentar uniformemente os valores das variáveis y_S com S em $\mathcal{S}_F$ de modo a não violar as restrições do programa dual. Esse aumento gradativo de alguns componentes de y pára quando a restrição $\sum y_S \leq c_e$ correspondente a alguma aresta externa e for satisfeita com igualdade. A aresta e é então acrescentada à floresta F.

O processo iterativo pára quando F não tem componentes ativos. O algoritmo devolve então uma $\mathcal{R}$-floresta minimal de F, ou seja, uma floresta F_1 tal que, para cada aresta e, a floresta $F_1 - e$ tem algum componente ativo.

A seguir, descrevemos o algoritmo formalmente, ainda que de maneira pouco detalhada. O algoritmo supõe que a instância $(G, c, \mathcal{R})$ é viável, ou seja, que cada conjunto em $\mathcal{R}$ está contido em algum componente de G.

Algoritmo MinFS-GW $(G, c, \mathcal{R})$

$F \leftarrow (V, \emptyset)$
para cada S em $\mathcal{S}$ faça $y_S \leftarrow 0$
enquanto $\mathcal{S}_F \neq \emptyset$ faça
    seja y' o vetor característico de $\mathcal{S}_F$
    seja θ o maior número tal que
        $\sum_{S\,:\,e\in\delta(S)} (y + \theta y')_S \leq c_e$ para cada aresta externa e

7 seja f uma aresta externa tal que $\sum_{S\,:\,f\in\delta(S)}(y+\theta y')_S = c_f$
8 $y \leftarrow y + \theta y'$
9 $F \leftarrow F + f$
10 $F_0 \leftarrow F$
11 seja F_1 uma $\mathcal{R}$-floresta minimal de F_0
12 devolva F_1

A cardinalidade da coleção $\mathcal{S}$ não é limitada por uma função polinomial de $|V|$ (por exemplo, $|\mathcal{S}|$ pode ser da ordem de $2^{|V|/2}$). Assim, o número de componentes do vetor y não será polinomial. É possível contornar essa dificuldade se registrarmos apenas os componentes não-nulos de y. O número total de tais componentes é limitado por um polinômio em $|V|$; de fato, o número de iterações não supera $|V|$, o número de componentes que assumem valores não-nulos em cada iteração é limitado por $|\mathcal{S}_F|$ e $|\mathcal{S}_F| < |V|$.[2] Vale a mesma observação para o vetor y' na linha 4, indexado por $\mathcal{S}$ e definido por $y'_S = 1$ se e somente se $S \in \mathcal{S}_F$.

O cálculo de θ nas linhas 5 e 6 pode ser organizado da seguinte maneira. Para cada vértice v, seja $d(v) := \sum_{S:v\in S} y_S$. Para cada aresta uv externa a F, defina $\theta_{uv} := \infty$ se uv liga dois componentes inativos de F, defina $\theta_{uv} := c_e - d(u) - d(v)$ se uv liga um componente ativo de F a um componente inativo, e defina $\theta_{uv} := \frac{1}{2}(c_e - d(u) - d(v))$ se uv liga dois componentes ativos. Finalmente, adote $\theta := \min_e \theta_e$, com o mínimo tomado sobre todas as arestas externas.

Deixamos a cargo do leitor a descrição polinomial do algoritmo MINFS-GW. É possível implementá-lo de modo que seu consumo de tempo seja $O(|V|^2 \log |V|)$ [GW95a].

Lema 5.3: *O algoritmo* MINFS-GW *admite uma implementação polinomial.* □

Análise do algoritmo

No início de cada iteração, F é uma floresta geradora de G. Ao final do processo iterativo, a floresta F_0 não tem componentes ativos e portanto é uma $\mathcal{R}$-floresta. A floresta F_1 que o algoritmo devolve também é uma $\mathcal{R}$-floresta.

[2] O número de componentes não-nulos de y é limitado por $2|V|-1$ (exercício 5.17).

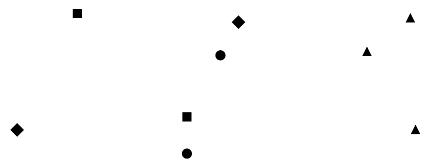


Figura 5.1: A figura representa uma instância $(G, c, \{R_1, R_2, R_3\})$ do MinFS. O grafo G é completo e tem 9 vértices. As arestas estão implícitas; o custo c_e de cada aresta e é a distância euclidiana entre os extremos de e. Os elementos de R_1, R_2 e R_3 estão representados por triângulos, losangos e quadrados.

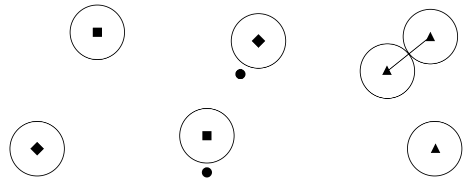


Figura 5.2: A figura mostra o início da segunda iteração do algoritmo MinFS-GW. Os círculos representam componentes ativos. O raio de cada círculo S é proporcional ao valor da variável dual y_S.

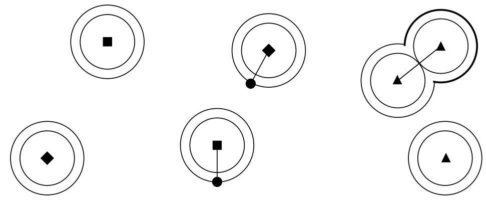


Figura 5.3: Situação no início da quarta iteração. As arestas da floresta F são indicadas por linhas sólidas. (Continua na próxima figura.)

Antes de delimitar $c(F_1)$, é preciso estabelecer uma relação fundamental entre F_1 e a coleção $\mathcal{S}_F$ dos componentes ativos de F em uma iteração arbitrária do algoritmo.

Lema 5.4: *No início de cada iteração, vale a desigualdade*

$$\sum_{S \in \mathcal{S}_F} |\delta_{F_1}(S)| \;\leq\; 2|\mathcal{S}_F| \,,$$

onde F_1 é a floresta de Steiner que o algoritmo devolve.

Demonstração: Digamos que o *grau* em F_1 de um componente S de F é o número de arestas de F_1 em $\delta(S)$, ou seja, $|\delta_{F_1}(S)|$. Os componentes de F de grau 1 em F_1 são todos ativos: para qualquer componente S de F,

$$\text{se } |\delta_{F_1}(S)| = 1 \text{ então } S \in \mathcal{S}_F \,. \tag{5.16}$$

Para provar essa afirmação, tome um componente S de F tal que $\delta_{F_1}(S)$ contém uma única aresta, digamos uw; ajuste a notação de modo que $u \in S$. Sejam U e W os conjuntos de vértices dos componentes de $F_1 - uw$ que contêm u e w respectivamente. Como uw é a única aresta de F_1 em $\delta(S)$, temos $U \subseteq S$ e $W \subseteq V \setminus S$. Como F_1 é uma $\mathcal{R}$-floresta minimal, existe R em $\mathcal{R}$ tal que

$$R \subseteq U \cup W, \quad R \cap U \neq \emptyset \quad \text{e} \quad R \cap W \neq \emptyset \,.$$

Segue daí que $R \cap S \neq \emptyset$ e $R \setminus S \neq \emptyset$. Assim, S está em $\mathcal{S}$ e portanto em $\mathcal{S}_F$. Isso conclui a prova de (5.16).

$\mathcal{Z}_F$ Seja $\mathcal{Z}_F$ o conjunto de todos os componentes inativos de F cujo grau em F_1 não é nulo, ou seja, todos os componentes inativos S para os quais $|\delta_{F_1}(S)| \geq 1$. Digamos que dois elementos S e S' de $\mathcal{S}_F \cup \mathcal{Z}_F$ são *adjacentes* se existe uma aresta de F_1 com um extremo em S e outro
H em S'. Esse conceito de adjacência define um grafo, digamos H, sobre o conjunto de vértices $\mathcal{S}_F \cup \mathcal{Z}_F$. Como F e F_1 são subgrafos de F_0, qualquer circuito em H corresponderia, de maneira natural, a um circuito em F_0, o que é impossível, pois F_0 é uma floresta. Portanto, H é uma floresta. Segue daí imediatamente que $\sum_{S \in V_H} |\delta_H(S)| = 2|E_H| \leq 2(|V_H| - 1)$, donde

$$\sum_{S \in \mathcal{S}_F \cup \mathcal{Z}_F} |\delta_{F_1}(S)| \;\leq\; 2(|\mathcal{S}_F| + |\mathcal{Z}_F| - 1) \,.$$

Em virtude de (5.16), temos que $|\delta_{F_1}(S)| \geq 2$ para cada S em $\mathcal{Z}_F$. Portanto,

$$\begin{aligned} \sum_{S \in \mathcal{S}_F} |\delta_{F_1}(S)| &= \sum_{S \in \mathcal{S}_F \cup \mathcal{Z}_F} |\delta_{F_1}(S)| - \sum_{S \in \mathcal{Z}_F} |\delta_{F_1}(S)| \\ &\leq 2(|\mathcal{S}_F| + |\mathcal{Z}_F| - 1) - 2|\mathcal{Z}_F| \\ &< 2|\mathcal{S}_F| \,. \end{aligned}$$

Poderíamos interpretar essa desigualdade dizendo que o grau médio dos vértices ativos do grafo H não é maior que 2. □

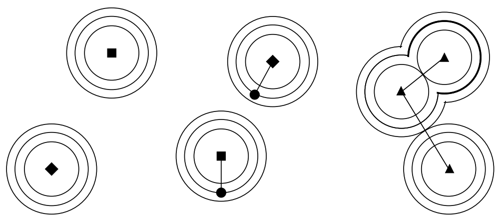


Figura 5.4: Situação no início da quinta iteração.

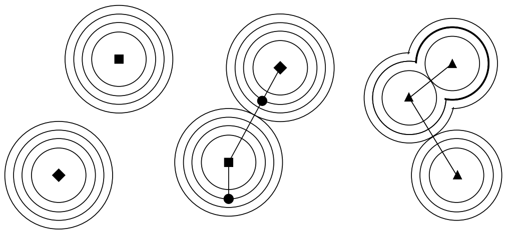


Figura 5.5: Início da sexta iteração.

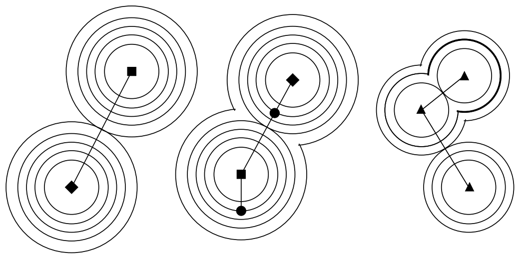


Figura 5.6: Início da sétima iteração.

Vamos mostrar agora que o custo da floresta de Steiner produzida pelo algoritmo difere do custo de uma floresta de Steiner ótima por um fator inferior a 2.

Teorema 5.5: *O algoritmo* MinFS-GW *é uma* 2*-aproximação para o* MinFS.

Demonstração: Como já observamos no início da presente seção, o subgrafo F_1 que o algoritmo devolve é uma $\mathcal{R}$-floresta. No início de cada iteração, vale a desigualdade

$$\textstyle\sum_{S \in \mathcal{S}} |\delta_{F_1}(S)|\, y_S \;\leq\; 2y(\mathcal{S})\,, \tag{5.17}$$

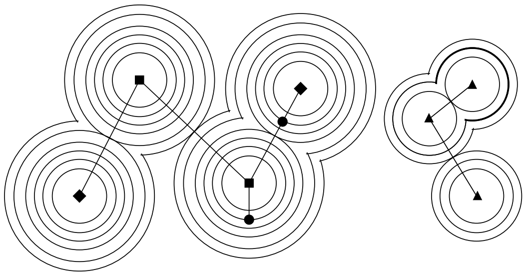


Figura 5.7: Fim da última iteração. As linhas indicam as arestas da floresta F_0.

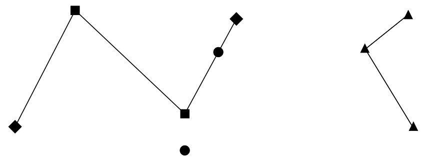


Figura 5.8: As linhas indicam a floresta final F_1.

como passamos a mostrar. É evidente que a desigualdade é válida no início da primeira iteração, quando $y = 0$. Suponha agora que a desigualdade seja válida no início de uma iteração qualquer. Durante a iteração, y_S é acrescido de θ se e somente se $S \in \mathcal{S}_F$. Portanto, o lado esquerdo de (5.17) é acrescido de

$$\textstyle\sum_{S \in \mathcal{S}_F} |\delta_{F_1}(S)|\theta$$

enquanto o lado direito é acrescido de $2|\mathcal{S}_F|\theta$. O lema 5.4 garante que o incremento do lado esquerdo não é maior que o do lado direito. Portanto a desigualdade (5.17) vale no início da iteração seguinte. Isso prova (5.17).

No fim do processo iterativo, o vetor y é viável no programa (5.11). Além disso, a condição (5.15) de folgas 1-aproximadas vale com F_0 no papel de F e portanto também com F_1 no papel de F:

$$\textstyle\sum_{S\,:\,e \in \delta(S)} y_S = c_e \text{ para cada } e \text{ em } F_1 \,.$$

Para mostrar que o algoritmo é uma 2-aproximação, resta apenas verificar que $c(F_1) \le 2\,\mathrm{opt}(G, c, \mathcal{R})$:

$$c(F_1) \quad = \quad \textstyle\sum_{e \in F_1} c_e$$

$$
\begin{aligned}
&= \textstyle\sum_{e\in F_1} \sum_{S\,:\,e\in\delta(S)} y_S \\
&= \textstyle\sum_{S\in\mathcal{S}} |\delta_{F_1}(S)|\, y_S && (5.18)\\
&\le 2\,y(\mathcal{S}) && (5.19)\\
&\le 2\,\mathrm{opt}(G,c,\mathcal{R})\ . && (5.20)
\end{aligned}
$$

A linha (5.18) segue da anterior por mera reorganização da soma. A desigualdade (5.19) segue de (5.17). Finalmente, a desigualdade (5.20) é conseqüência da delimitação inferior (5.12). □

Exercícios

5.1 Utilize o lema de Farkas para mostrar que exatamente um dos problemas $\mathrm{RP}(A,b,y)$ e $\mathrm{RD}(A,b,y)$ é viável.

5.2 Seja y um vetor em $Y(A,c)$ e y' um vetor tal que $y+\theta y'$ está em $Y(A,c)$ para todo θ positivo. Mostre que y' está em $Y(A,0)$. Conclua que o vetor y' devolvido na linha 5 dos métodos PRIMAL-DUAL clássico e APROXIMAÇÃO-PRIMAL-DUAL é um vetor de inviabilidade de $\mathrm{P}(A,b,c)$ e que $\mathrm{D}(A,c,b)$ é ilimitado.

5.3 Como θ pode ser calculado na linha 6 dos métodos PRIMAL-DUAL clássico e APROXIMAÇÃO-PRIMAL-DUAL? Conclua que θ é positivo.

5.4 Como determinar, na linha 4 dos métodos PRIMAL-DUAL clássico e APROXIMAÇÃO-PRIMAL-DUAL, se $y+\theta y'$ está em $Y(A,c)$ para todo θ positivo?

5.5 Considere o método APROXIMAÇÃO-PRIMAL-DUAL. Mostre que um dos problemas restritos aproximados é viável. Ambos problemas podem ser simultaneamente viáveis?

5.6 Seja y um vetor em $Y(A,c)$ e x uma solução de $\mathrm{RAP}(A,b,y,\alpha,\beta)$. Mostre que x e y têm folgas α-aproximadas no primal e β-aproximadas no dual. Conclua que $\alpha\, cx \le \beta\, y\, b$.

5.7 Verifique que o método PRIMAL-DUAL clássico é o método APROXIMAÇÃO-PRIMAL-DUAL para $\alpha = 1 = \beta$.

5.8 Formule o **problema do caminho mínimo** como um problema da transversal mínima equivalente.

> **Problema** MINPATH (G, s, t, c): *Dados um grafo G, dois vértices s e t de G e um custo c_e em $\mathbb{Q}_\geq$ para cada aresta e, encontrar um caminho P de s a t de custo mínimo.*

5.9 Formule o **problema da árvore geradora mínima** como um problema da transversal mínima equivalente.

> **Problema** MST (G, c): *Dados um grafo G e um custo c_e em $\mathbb{Q}_\geq$ para cada aresta e, encontrar uma árvore geradora de custo mínimo.*

5.10 Dados um conjunto T de vértices e um conjunto F de arestas de um grafo G, dizemos que F é uma T-**junção** se T é o conjunto dos vértices de grau ímpar do grafo $G[F]$. Formule o **problema da T-junção mínima** como um problema da transversal mínima equivalente.

> **Problema** MINTJ (G, T, c): *Dados um grafo G, um subconjunto T de V_G tal que $|T|$ é par e um custo c_e em $\mathbb{Q}_\geq$ para cada aresta e, encontrar uma T-junção F que minimize $c(F)$.*

5.11 Formule o **problema do corte mínimo** como um problema da transversal mínima equivalente.

> **Problema** MINCUT (G, c): *Dados um grafo G e um custo c_e em $\mathbb{Q}_\geq$ para cada aresta e, encontrar um corte R que minimize $c(R)$.*

5.12 Para cada um dos problemas nos quatro exercícios anteriores, descreva um algoritmo que, dados um grafo G e um subconjunto T de E_G, decide se T é uma transversal da respectiva coleção $\mathcal{S}$. O consumo de tempo de cada algoritmo deve ser polinomial em $\langle G \rangle$ e não deve depender de $|\mathcal{S}|$.

5.13 Mostre que o programa linear (5.5) é uma relaxação linear do MINTC $(E, \mathcal{S}, c)$.

5.14 Formule o problema da cobertura mínima por vértices (MINCV) como um problema da transversal mínima MINTC $(E, \mathcal{S}, c)$ equivalente. Mostre que o algoritmo MINTC-BE é uma 2-aproximação polinomial para o MINCV.

5.15 Formule o problema do multicorte mínimo (MINMCUT) como um problema da transversal mínima MINTC $(E, \mathcal{S}, c)$ equivalente. Mostre que, se as linhas a seguir são inseridas entre as linhas 9 e 10 do algoritmo MINTC-BE, então o algoritmo resultante é uma 2-aproximação polinomial para o MINMCUT quando restrito a árvores [GVY97].

```
9a  sejam f_1, ..., f_k os elementos de J, na ordem em que
9b      foram incluídos em J
9c  para i de k até 1 faça
9d      se J \ {f_i} é uma transversal então J ← J \ {f_i}
```

5.16 No problema MINFS$(G, c, \mathcal{R})$, não há perda de generalidade se supusermos que G é completo e os custos satisfazem a desigualdade triangular. Discuta essa afirmação.

5.17 Uma coleção $\mathcal{L}$ de subconjuntos de um conjunto finito X é **laminar** se $L_1 \subseteq L_2$ ou $L_1 \supseteq L_2$ ou $L_1 \cap L_2 = \emptyset$ para cada par (L_1, L_2) de elementos de $\mathcal{L}$. Mostre que se $\mathcal{L}$ é laminar então $|\mathcal{L}| \leq 2|X| - 1$. Mostre que no algoritmo MINFS-GW a coleção $\{S \in \mathcal{S} : y_S > 0\}$ é laminar.

5.18 Mostre que a razão de aproximação do algoritmo MINFS-GW $(G, c, \mathcal{R})$ é $2 - 2/n$, onde $n := |V_G|$.

5.19 Exiba uma instância do problema MINFS para a qual o algoritmo MINFS-GW produz uma árvore de Steiner de custo essencialmente igual ao dobro do ótimo.

5.20 Aplique o algoritmo MINFS-GW ao problema da árvore geradora mínima (exercício 5.9). Mostre que ele se comporta como o algoritmo de Kruskal [CLR92] (e portanto produz uma árvore geradora mínima).

5.21 Escreva o algoritmo MINFS-GW de maneira mais detalhada. Em particular, escreva uma implementação que, para uma instância $(G, c, \mathcal{R})$, consuma tempo polinomial em $\langle G \rangle$.

Notas bibliográficas

Este capítulo foi escrito com base nos livros de Schrijver [Sch86], Papadimitriou e Steiglitz [PS82], Hochbaum [Hoc97] e no artigo de Goemans e Williamson [GW95a].

O livro de Papadimitriou e Steiglitz [PS82] apresenta diversos algoritmos combinatórios clássicos que podem ser entendidos como uma mímica do método primal-dual, incluindo o algoritmo de Dijkstra [Dij59] para encontrar caminhos de custo mínimo, o algoritmo de Ford e Fulkerson [FF56] para encontrar um fluxo máximo, algoritmos primal-dual para o problema do fluxo de custo mínimo, o método húngaro de Kuhn [Kuh55] para encontrar um emparelhamento de custo máximo em um grafo bipartido, e o algoritmo de Edmonds [Edm65a] para encontrar um emparelhamento de custo máximo em um grafo.

O primeiro algoritmo de aproximação genuinamente primal-dual, ou seja, o primeiro a usar uma solução dual viável para obter, iterativamente, uma solução aproximada do problema primal, foi o algoritmo de Bar-Yehuda e Even [BYE81] para o problema da cobertura mínima por vértices. Trata-se, essencialmente, do algoritmo descrito na seção 5.3. A análise dessa abordagem aparece em um artigo de Hochbaum [Hoc82], que obteve um algoritmo com a mesma razão de aproximação usando o método dual.

A primeira 2-aproximação para o problema da floresta de Steiner foi obtida por Agrawal, Klein e Ravi [AKR95]. Esse trabalho introduziu modificações poderosas no método primal-dual básico, e com isso deu um grande impulso às pesquisas sobre problemas de projetos de redes (*network design problems*), como atestam os diversos algoritmos de aproximação primal-dual obtidos por essa época [AG94, GGW98, GGP$^+$94, GW95a, Rav94, RW95, WGMV95]. Segundo Goemans e Williamson [Hoc97], esse trabalho de Agrawal, Klein e Ravi foi o primeiro a fazer uso altamente sofisticado do método primal-dual para projetar algoritmos de aproximação.

O problema da floresta de Steiner pode ser classificado como um problema de projeto de redes. Esse tipo de problema consiste em exigir um certo grau de conexão entre certos pares de vértices de um grafo. O conceito de funções próprias unifica vários desses problemas: o problema da floresta de Steiner, o problema do caminho mínimo, o problema da conexão ponto-a-ponto, o problema da T-junção mínima, o problema do emparelhamento perfeito de peso mínimo, etc. Usando apenas as propriedades que caracterizam funções próprias, Goemans e Williamson [GW95a, Hoc97] mostraram que todos esses problemas admitem uma 2-aproximação polinomial.

O melhor resultado que se conhece para o problema da árvore de Steiner é uma 1,55-aproximação que não se baseia em programação linear, obtido recentemente por Robins e Zelikovsky [RZ00].

Experimentos computacionais realizados com alguns algoritmos de aproximação obtidos pelo método primal-dual mostram que eles têm bom desempenho na prática. Segundo Goemans e Williamson [WG94], a 2-aproximação para o problema do emparelhamento perfeito de peso mínimo em que os custos satisfazem a desigualdade triangular encontra soluções a 2% do ótimo na maioria dos casos (em mais de 1400 experimentos, o resultado nunca foi pior do que 4%). Os mesmos autores relatam bons resultados práticos para o problema da floresta de Steiner e alguns outros problemas de redes.

Gabow, Goemans e Williamson [GGW98] obtiveram uma implementação mais eficiente de vários algoritmos para problemas de projetos de redes, incluindo o problema da floresta de Steiner. Finalmente, há vários trabalhos recentes [CRW01, GVY97, JMVW99, JV00, GW98, RV99] a respeito de métodos de aproximação primal-dual, indicando que se trata de um tema de interesse atual e aparentemente muito promissor.

Capítulo 6

Algoritmos Probabilísticos

Neste capítulo vamos supor que temos à nossa disposição um gerador de *bits* aleatórios (*random bits generator*), ou seja, um algoritmo ideal Rand que recebe um número racional ρ no intervalo fechado $[0, 1]$, devolve 1 com probabilidade ρ e 0 com probabilidade $1 - \rho$. Existem implementações aproximadas do algoritmo Rand (veja Knuth [Knu98]). Rand

Um algoritmo que utiliza Rand é chamado de **probabilístico** (*randomized*). O comportamento de um algoritmo probabilístico depende não apenas da instância do problema mas também dos números devolvidos por Rand. Dizemos que um algoritmo probabilístico é **polinomial** se existe um polinômio p tal que, para toda instância I do problema, o número de chamadas ao algoritmo Rand e o consumo de tempo das demais operações são limitados por $p(\langle I \rangle)$.

Vamos generalizar a definição de algoritmos de aproximação (seção 1.2) para englobar algoritmos probabilísticos. Considere um algoritmo probabilístico A para um problema de otimização. Para qualquer instância I do problema, seja X_I a variável aleatória (apêndice D) cujo valor é o valor da solução produzida por A para esta instância[1]. Dizemos que A é uma **α-aproximação probabilística** para o problema em questão se $\mathbf{E}[X_I] \geq \alpha \, \mathrm{opt}(I)$ no caso de problema de maximização e $\mathbf{E}[X_I] \leq \alpha \, \mathrm{opt}(I)$ no caso de problema de minimização. No presente capítulo, α é um número que não depende de I, embora em geral α possa

[1] Os algoritmos abordados neste capítulo, para cada instância do problema, provocam um número fixo t de chamadas de Rand, que não depende dos valores devolvidos por Rand, sendo que a i-ésima chamada de Rand tem como parâmetro sempre um mesmo número, digamos ρ_i. Denotando por Ω o conjunto $\{0, 1\}$, seja $\mathcal{P}_i$ a medida de probabilidade sobre Ω dada por $\mathcal{P}_i(1) = \rho_i$ e $\mathcal{P}_i(0) = 1 - \rho_i$. O espaço de probabilidade no qual X_I está definida neste caso é o espaço produto $(\Omega, \mathcal{P}_1) \times \cdots \times (\Omega, \mathcal{P}_t)$.

ser uma função de I. Dizemos que α é uma **razão de aproximação esperada** de A.

Vamos mostrar que, em certas condições, uma α-aproximação probabilística A pode ser usada para produzir, com alta probabilidade, soluções aproximadas do problema. Suponha que o problema é de minimização e que a variável aleatória X_I não assume valores negativos (esse é o caso na grande maioria dos problemas combinatórios). Para cada ε positivo, a desigualdade de Markov (lema D.1, apêndice D) garante que

$$\mathbf{Pr}[X_I \geq (\alpha+\varepsilon)\mathrm{opt}(I)] \leq \frac{\mathbf{E}[X_I]}{(\alpha+\varepsilon)\mathrm{opt}(I)} \leq \frac{\alpha\,\mathrm{opt}(I)}{(\alpha+\varepsilon)\mathrm{opt}(I)} = \frac{\alpha}{\alpha+\varepsilon}\,.$$

Para um dado λ no intervalo aberto $(0,1)$, seja $k := \lceil \log_\beta \lambda \rceil$, onde $\beta = \frac{\alpha}{\alpha+\varepsilon}$. Suponha que A é aplicado k vezes à instância I e escolha o resultado de menor valor. Se Y_I é esse menor valor[2], então $\mathbf{Pr}[Y_I \geq (\alpha+\varepsilon)\mathrm{opt}(I)] \leq \lambda$. Para λ pequeno, este esquema produz, portanto, com probabilidade alta (pelo menos $1-\lambda$), uma solução viável com valor menor que $(\alpha+\varepsilon)\mathrm{opt}(I)$.

O mesmo raciocínio pode ser aplicado a problemas de maximização, desde que se conheça uma delimitação superior para o valor ótimo do problema.

6.1 Satisfatibilidade máxima

Vamos usar o problema da satisfatibilidade máxima para exemplificar algoritmos probabilísticos. Para definir o problema, precisamos introduzir alguma notação.

Suponha dado um conjunto finito V de objetos que chamamos **variáveis**. Uma **cláusula booleana** sobre V, ou simplesmente **cláusula**, é um par ordenado de subconjuntos de V disjuntos, um dos quais pelo menos não é vazio. Se C é uma cláusula, denotamos o primeiro componente de C por C_1 e o segundo por C_0. Em terminologia tradicional, C_0 é o conjunto das variáveis "complementadas" e C_1 é o conjunto das variáveis "não-complementadas". O **número de variáveis** em uma cláusula C é $|C_1| + |C_0|$.

[2] Y_I é uma variável aleatória. Denotando por $(\Omega', \mathcal{P}')$ o espaço de probabilidade de X_I, o espaço no qual Y_I está definida é o produto $(\Omega', \mathcal{P}') \times \cdots \times (\Omega', \mathcal{P}')$ de k cópias de $(\Omega', \mathcal{P}')$.

Uma **valoração** das variáveis de V é um vetor x indexado por V com valores em $\{0,1\}$. Uma valoração x **satisfaz** uma cláusula C se $x_v = 1$ para algum v em C_1 ou $x_v = 0$ para algum v em C_0.

Vejamos um exemplo. Suponha que $V = \{a,b,c,d\}$ e sejam C, D e E as cláusulas $(\{a,d\},\{b\})$, $(\{c\},\{a,b\})$ e $(\emptyset,\{b,c,d\})$. Na notação tradicional, a coleção $\{C,D,E\}$ seria denotada por $\{a \vee d \vee \bar{b}, c \vee \bar{a} \vee \bar{b}, \bar{b} \vee \bar{c} \vee \bar{d}\}$.

O **problema da satisfatibilidade máxima** (*maximum satisfiability problem*), denotado pela sigla MAXSAT, consiste no seguinte:

> **Problema** MAXSAT $(V, \mathcal{C})$: *Dada uma coleção $\mathcal{C}$ de cláusulas sobre um conjunto V de variáveis, encontrar uma valoração x de V que satisfaça o maior número possível de cláusulas de $\mathcal{C}$.*

O problema é NP-difícil mesmo quando cada cláusula em $\mathcal{C}$ tem duas variáveis [GJ79].

Algoritmo de Johnson

Eis um algoritmo probabilístico para o problema MAXSAT. O algoritmo é atribuído a Johnson [Joh74]. Ele é tão simples que ignora $\mathcal{C}$:

Algoritmo MAXSAT-JOHNSON (V)

```
1  para cada v em V faça
2      ẋ_v ← RAND (1/2)
3  devolva ẋ
```

No algoritmo acima, o número de chamadas a RAND e o consumo de tempo das demais operações é $O(|V|)$. Seja $X_{\mathcal{C}}$ a variável aleatória cujo valor é o número de cláusulas de $\mathcal{C}$ satisfeitas por uma valoração produzida por MAXSAT-JOHNSON (V). O espaço de probabilidade de $X_{\mathcal{C}}$ é o espaço produto $(\Omega, \mathcal{P}) \times \cdots \times (\Omega, \mathcal{P})$, onde $(\Omega, \mathcal{P})$ aparece $|V|$ vezes no produto, Ω é o conjunto $\{0,1\}$ e $\mathcal{P}$ é a medida de probabilidade sobre Ω dada por $\mathcal{P}(1) = \mathcal{P}(0) = \frac{1}{2}$. $X_{\mathcal{C}}$

> **Teorema 6.1**: *Se $(V, \mathcal{C})$ é uma instância do problema* MAXSAT *na qual cada cláusula tem pelo menos k variáveis e $X_{\mathcal{C}}$ é a variável aleatória acima definida, então*
>
> $$\mathbf{E}[X_{\mathcal{C}}] \;\geq\; \left(1 - \frac{1}{2^k}\right) \operatorname{opt}(V, \mathcal{C})\,.$$

Demonstração: Para cada cláusula C, defina uma variável aleatória Z_C da seguinte maneira: $Z_C := 1$ se a valoração produzida pelo algoritmo satisfaz C e $Z_C := 0$ em caso contrário. Como C tem pelo menos k variáveis e no algoritmo $\dot{x}_v$ é 0 com probabilidade $1/2$, a probabilidade do evento $[Z_C{=}0]$ é no máximo $1/2^k$ e a probabilidade do evento $[Z_C{=}1]$ é pelo menos $1-1/2^k$. Portanto, como $X_{\mathcal{C}}$ é a soma das variáveis aleatórias Z_C,

$$\mathbf{E}[X_{\mathcal{C}}] \;=\; \mathbf{E}[\textstyle\sum_C Z_C] \;=\; \textstyle\sum_C \mathbf{E}[Z_C] \;\geq\; (1-\dfrac{1}{2^k})|\mathcal{C}|$$

e concluímos a prova do teorema, já que $\mathrm{opt}(V,\mathcal{C}) \leq |\mathcal{C}|$. □

Como todas as cláusulas em $\mathcal{C}$ têm pelo menos uma variável, podemos aplicar o teorema acima a $(V,\mathcal{C})$ com $k=1$ e concluir o seguinte resultado.

Teorema 6.2: *O algoritmo* MAXSAT-JOHNSON *é uma* 0,5-*aproximação probabilística polinomial para o* MAXSAT.

Algoritmo do arredondamento probabilístico

Descrevemos a seguir um algoritmo de aproximação para o problema MAXSAT que envolve o "arredondamento probabilístico" (*randomized rounding*) de uma solução ótima de um programa linear. O **arredondamento probabilístico** de um número ρ do intervalo aberto $(0,1)$ consiste em arredondar ρ "para cima" com probabilidade ρ e "para baixo" com probabilidade $1-\rho$.

Considere o seguinte problema de programação linear: dada uma coleção $\mathcal{C}$ de cláusulas sobre um conjunto V, encontrar um par (x,z) de vetores, com x indexado por V e z indexado por $\mathcal{C}$, que

$$\begin{array}{lrcll}
\text{maximize} & \sum_{C\in\mathcal{C}} z_C & & & \\
\text{sob as restrições} & & & & \\
\sum_{v\in C_0}(1-x_v)+\sum_{v\in C_1} x_v & \geq & z_C & & \text{para cada } C \text{ em } \mathcal{C}\,, \\
0 \;\leq & z_C & \leq & 1 & \text{para cada } C \text{ em } \mathcal{C}\,, \\
0 \;\leq & x_v & \leq & 1 & \text{para cada } v \text{ em } V\,.
\end{array} \tag{6.1}$$

Note a seguinte relação entre este programa linear e o problema MAXSAT. Suponha que temos uma solução ótima x^* do MAXSAT $(V,\mathcal{C})$. Para cada cláusula C em $\mathcal{C}$, faça $z^*_C := 1$ se x^* satisfaz C e $z^*_C := 0$ em caso contrário. É claro então que o par (x^*, z^*) é uma solução viável de (6.1) e o valor dessa solução é igual ao número de cláusulas de $\mathcal{C}$ satisfeitas por x^*. Portanto, o programa linear (6.1) é uma relaxação do problema MaxSat $(V, \mathcal{C})$ e

$$\textstyle\sum_C \hat{z}_C \geq \mathrm{opt}(V, \mathcal{C})\,, \tag{6.2}$$

para qualquer solução ótima $(\hat{x}, \hat{z})$ de (6.1).

O seguinte algoritmo de arredondamento probabilístico, desenvolvido por Goemans e Williamson [GW94], é um algoritmo de aproximação para o MaxSat.

Algoritmo MaxSat-GW $(V, \mathcal{C})$

```
1  seja (x̂, ẑ) uma solução ótima racional de (6.1)
2  para cada v em V faça
3      ẋ_v ← Rand (x̂_v)
4  devolva ẋ
```

O algoritmo acima pode ser implementado de modo que a execução da linha 1 consuma tempo limitado por um polinômio em $|V|$ e $|\mathcal{C}|$, e produza uma solução ótima racional $(\hat{x}, \hat{z})$ com o tamanho de $\hat{x}$ e $\hat{z}$ limitados por um polinômio em $|V|$ e $|\mathcal{C}|$ (fato C.4, apêndice C). Como, além disso, o número de chamadas a Rand é $|V|$, o algoritmo MaxSat-GW é um algoritmo probabilístico polinomial.

Teorema 6.3: *Se* $(V, \mathcal{C})$ *é uma instância do problema* MaxSat *na qual cada cláusula tem no máximo* k *variáveis, com* $k \geq 1$, *e* $X_{\mathcal{C}}$ *é a variável aleatória*[3] *cujo valor é o número de cláusulas de* $\mathcal{C}$ *satisfeitas por uma valoração produzida pelo algoritmo* MaxSat-GW $(V, \mathcal{C})$, *então*

$$\mathbf{E}[X_{\mathcal{C}}] \;\geq\; \Big(1 - (1 - \frac{1}{k})^k\Big)\, \mathrm{opt}(V, \mathcal{C})\,.$$

Demonstração: Para cada cláusula C, defina Z_C da maneira usual: Z_C vale 1 se a valoração produzida pelo algoritmo satisfaz C e 0 em caso contrário. A probabilidade de que Z_C seja 0 é o produto de termos da forma $\hat{x}_v$, para v em C_0, e $(1 - \hat{x}_v)$, para v em C_1. Se t é o número de

[3] O espaço de probabilidade é $(\Omega, \mathcal{P}_1) \times \cdots \times (\Omega, \mathcal{P}_{|V|})$, onde $\Omega = \{0, 1\}$ e $\mathcal{P}_i$ é dado por $\mathcal{P}_i(1) = \hat{x}_i$ e $\mathcal{P}_i(0) = 1 - \hat{x}_i$ para cada i em V.

variáveis de C então $t \geq 1$ e

$$\begin{aligned}
\mathbf{Pr}[Z_C{=}1] &= 1 - \textstyle\prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v) \\
&\geq 1 - ((t - \hat{z}_C)/t)^t && (6.3) \\
&= 1 - (1 - \hat{z}_C/t)^t \\
&\geq (1 - (1 - 1/t)^t)\hat{z}_C && (6.4) \\
&\geq (1 - (1 - 1/k)^k)\hat{z}_C\ . && (6.5)
\end{aligned}$$

Para justificar (6.3), note que a média geométrica de números não-negativos não ultrapassa a sua média aritmética. Disso temos que

$$\begin{aligned}
\textstyle\prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v) &\leq \left(\left(\textstyle\sum_{v \in C_0} \hat{x}_v + \sum_{v \in C_1} (1 - \hat{x}_v)\right)/t\right)^t \\
&= \left(\left(t - \textstyle\sum_{v \in C_0} (1 - \hat{x}_v) - \sum_{v \in C_1} \hat{x}_v\right)/t\right)^t \\
&\leq ((t - \hat{z}_C)/t)^t\ .
\end{aligned}$$

Para verificar (6.4), considere a função $f(z) := 1 - (1 - z/t)^t$. Observe que f é côncava no intervalo fechado $[0, 1]$, pois $f'(z) \geq 0$ e $f''(z) \leq 0$ nesse intervalo. Então, como $f(0) = 0$, temos que $f(z) \geq z\, f(1)$, que se traduz em (6.4). Finalmente, (6.5) vale pois $(1 - 1/t)^t$ é crescente para $t \geq 1$ e, por hipótese, $t \leq k$.

Podemos agora calcular o valor esperado de $X_{\mathcal{C}}$:

$$\begin{aligned}
\mathbf{E}[X_{\mathcal{C}}] &= \mathbf{E}[\textstyle\sum_C Z_C] \\
&= \textstyle\sum_C \mathbf{E}[Z_C] \\
&= \textstyle\sum_C \mathbf{Pr}[Z_C{=}1] \\
&\geq \textstyle\sum_C (1 - (1 - 1/k)^k)\hat{z}_C \\
&\geq (1 - (1 - 1/k)^k)\,\mathrm{opt}(V, \mathcal{C})\ ,
\end{aligned}$$

onde a última desigualdade segue de (6.2). □

Como $(1 - 1/k)^k$ é não passa de $1/e$ para todo $k \geq 1$, onde e é a base dos logaritmos naturais, temos que $(1 - 1/k)^k < 1/e < 0{,}37$. Desse fato, e do teorema anterior, é imediato o seguinte resultado.

Teorema 6.4: *O algoritmo* MAXSAT-GW *é uma* 0,63*-aproximação probabilística polinomial para o* MAXSAT. □

Algoritmo combinado

O teorema 6.1 permite deduzir uma razão de aproximação melhor do algoritmo MAXSAT-JOHNSON quando todas as cláusulas têm pelo menos duas variáveis. Já para o algoritmo MAXSAT-GW obtêm-se resultados melhores quando as cláusulas têm no máximo duas variáveis, pelo teorema 6.3. Parece razoável, portanto, combinar os dois algoritmos. O algoritmo abaixo deve-se a Goemans e Williamson [GW94] também.

Algoritmo MAXSAT-COMBINADO-GW $(V, \mathcal{C})$

$\dot{x} \leftarrow$ MAXSAT-JOHNSON (V)
$\ddot{x} \leftarrow$ MAXSAT-GW $(V, \mathcal{C})$
seja $\dot{s}$ o número de cláusulas de $\mathcal{C}$ satisfeitas por $\dot{x}$
seja $\ddot{s}$ o número de cláusulas de $\mathcal{C}$ satisfeitas por $\ddot{x}$
se $\dot{s} \geq \ddot{s}$
    então devolva $\dot{x}$
    senão devolva $\ddot{x}$

Teorema 6.5: *O algoritmo* MAXSAT-COMBINADO-GW *é uma 0,75-aproximação probabilística polinomial para o* MAXSAT.

Demonstração: Seja $\mathcal{C}_k$ a coleção das cláusulas de $\mathcal{C}$ que têm exatamente k variáveis. Sejam $X_{\mathcal{C}}$, $\dot{X}_{\mathcal{C}}$ e $\ddot{X}_{\mathcal{C}}$ variáveis aleatórias cujos valores são o número de cláusulas de $\mathcal{C}$ satisfeitas por uma valoração produzida por MAXSAT-COMBINADO-GW $(V, \mathcal{C})$, MAXSAT-JOHNSON (V) e MAXSAT-GW $(V, \mathcal{C})$, respectivamente. Note que $X_{\mathcal{C}} \geq \frac{1}{2}(\dot{X}_{\mathcal{C}} + \ddot{X}_{\mathcal{C}})$. Se refizermos as partes apropriadas das demonstrações dos teoremas 6.1 e 6.3 temos

$$\begin{aligned}
\mathbf{E}[X_{\mathcal{C}}] &\geq \mathbf{E}[\tfrac{1}{2}(\dot{X}_{\mathcal{C}} + \ddot{X}_{\mathcal{C}})] \\
&= \tfrac{1}{2}(\mathbf{E}[\dot{X}_{\mathcal{C}}] + \mathbf{E}[\ddot{X}_{\mathcal{C}}]) \\
&\geq \tfrac{1}{2}\textstyle\sum_k \sum_{C \in \mathcal{C}_k} ((1 - 2^{-k}) + (1 - (1 - k^{-1})^k)\hat{z}_C) \\
&\geq \tfrac{1}{2}\textstyle\sum_k \sum_{C \in \mathcal{C}_k} (1 - 2^{-k} + 1 - (1 - k^{-1})^k)\hat{z}_C \\
&\geq \tfrac{1}{2}\textstyle\sum_k \sum_{C \in \mathcal{C}_k} \tfrac{3}{2}\,\hat{z}_C \qquad (6.6) \\
&\geq \tfrac{3}{4}\,\mathrm{opt}(V, \mathcal{C})\,. \qquad (6.7)
\end{aligned}$$

onde a desigualdade (6.6) pode ser concluída analisando-se os casos $k = 1$, $k = 2$ e $k \geq 3$ e (6.7) vale em virtude de (6.2). □

6.2 Desaleatorização

Em muitos casos, é possível converter um algoritmo de aproximação probabilístico em um determinístico com uma razão de aproximação igual à razão esperada do algoritmo probabilístico. Isso pode ser feito através do *método das esperanças condicionais* [AS92, ES74, Spe87]. Para aplicá-lo, precisamos saber calcular eficientemente certas esperanças condicionais, que dependem do problema em foco. É esse o cerne do processo de desaleatorização de um algoritmo probabilístico por esse método. Ilustramos o método aplicando-o ao algoritmo MaxSat-Johnson. Obtemos como resultado a seguinte 0,5-aproximação polinomial determinística, que utiliza um procedimento EspCond, detalhado posteriormente.

Algoritmo MaxSat-Johnson-Desaleatorizado $(V, \mathcal{C})$

$\mathcal{D} \leftarrow \mathcal{C}$
para cada v em V faça
    se EspCond $(v, 1, \mathcal{D}) \geq$ EspCond $(v, 0, \mathcal{D})$
        então $\dot{x}_v \leftarrow 1$
                para cada C em $\mathcal{D}$ faça
                    se $v \in C_1$
                        então $\mathcal{D} \leftarrow \mathcal{D} \setminus \{C\}$
                        senão $C_0 \leftarrow C_0 \setminus \{v\}$
        senão $\dot{x}_v \leftarrow 0$
                para cada C em $\mathcal{C}$ faça
                    se $v \in C_0$
                        então $\mathcal{D} \leftarrow \mathcal{D} \setminus \{C\}$
                        senão $C_1 \leftarrow C_1 \setminus \{v\}$
devolva $\dot{x}$

No algoritmo acima, $\mathcal{D}$ é um multiconjunto (um conjunto em que cada elemento aparece com uma certa multiplicidade). No início de cada iteração, $\mathcal{D}$ contém, além de pares $\{\emptyset, \emptyset\}$, que chamaremos de **pseudo-cláusulas**, apenas as cláusulas de $\mathcal{C}$ que a "valoração parcial" $\dot{x}$ do algoritmo ainda não satisfez. Ademais, de cada cláusula de $\mathcal{C}$ em $\mathcal{D}$, foram eliminadas as variáveis cujos valores $\dot{x}$ já fixou.

O procedimento auxiliar EspCond recebe uma variável v, um elemento i de $\{0, 1\}$ e um multiconjunto $\mathcal{D}$ de cláusulas e pseudo-cláusulas. Seja $X_{\mathcal{D}}$ a variável aleatória cujo valor é o número de cláusulas em $\mathcal{D}$, levando em conta a multiplicidade, satisfeitas por uma valoração produzida pelo algoritmo MaxSat-Johnson (V). Se $i = 1$ então o procedimento devolve o valor esperado de $X_{\mathcal{D}}$ sob a condição que $\dot{x}_v$=1 no algoritmo

MAXSAT-JOHNSON. Este valor esperado é denotado por $\mathbf{E}[X_{\mathcal{D}}|\dot{x}_v{=}1]$. Se $i = 0$ então o procedimento devolve $\mathbf{E}[X_{\mathcal{D}}|\dot{x}_v{=}0]$, que é definido de maneira análoga.

Procedimento ESPCOND $(v, i, \mathcal{D})$

```
esp ← 0
para cada C em D faça
    k ← |C_1| + |C_0|
    se v ∈ C_i
        então esp ← esp + 1
        senão se v ∈ C_{1−i}
                  então esp ← esp + (1 − 2^{−k+1})
                  senão esp ← esp + (1 − 2^{−k})
devolva esp
```

Teorema 6.6: *O algoritmo* MAXSAT-JOHNSON-DESALEATORIZADO *é uma* 0,5*-aproximação polinomial para o* MAXSAT.

Demonstração: Seja $X_{\mathcal{C}}$ a variável aleatória cujo valor é o número de cláusulas em $\mathcal{C}$ satisfeitas por uma valoração produzida pelo algoritmo MAXSAT-JOHNSON (V). Como, no algoritmo MAXSAT-JOHNSON, $\dot{x}_v = 1$ com probabilidade $1/2$ e $\dot{x}_v = 0$ com probabilidade $1/2$, temos que $\mathbf{E}[X_{\mathcal{C}}] = \frac{1}{2}\mathbf{E}[X_{\mathcal{C}}|\dot{x}_v{=}1] + \frac{1}{2}\mathbf{E}[X_{\mathcal{C}}|\dot{x}_v{=}0]$, e disso concluímos que $\mathbf{E}[X_{\mathcal{C}}]$ não ultrapassa o maior entre as duas esperanças condicionais, $\mathbf{E}[X_{\mathcal{C}}|\dot{x}_v{=}1]$ e $\mathbf{E}[X_{\mathcal{C}}|\dot{x}_v{=}0]$. Logo, $\mathbf{E}[X_{\mathcal{C}}] \leq \mathbf{E}[X_{\mathcal{C}}|\dot{x}_{v_1}]$, onde o último termo denota o valor esperado de $X_{\mathcal{C}}$ sob a condição que, no algoritmo MAXSAT-JOHNSON, atribua-se a $\dot{x}_{v_1}$ o mesmo valor atribuído no algoritmo MAXSAT-JOHNSON-DESALEATORIZADO. Uma vez fixado o valor de $\dot{x}_{v_1}$, o mesmo raciocínio vale para as demais variáveis e, disso, concluímos que

$$\mathbf{E}[X_{\mathcal{C}}] \leq \mathbf{E}[X_{\mathcal{C}}|\dot{x}_{v_1}] \leq \mathbf{E}[X_{\mathcal{C}}|\dot{x}_{v_1}, \dot{x}_{v_2}] \leq \ldots \leq \mathbf{E}[X_{\mathcal{C}}|\dot{x}_{v_1}, \ldots, \dot{x}_{v_n}]\,.$$

O último termo é o número de cláusulas de $\mathcal{C}$ satisfeitas pela valoração $\dot{x}$ produzida pelo algoritmo MAXSAT-JOHNSON-DESALEATORIZADO. Esse número é pelo menos $0{,}5\,\mathrm{opt}(V, \mathcal{C})$, pois $\mathbf{E}[X_{\mathcal{C}}] \geq 0{,}5\,\mathrm{opt}(V, \mathcal{C})$.

Quanto ao tempo de execução do algoritmo, basta observar que o procedimento ESPCOND $(v, i, \mathcal{D})$ consome tempo $O(|V|)$. Disso, é fácil concluir que o algoritmo MAXSAT-JOHNSON-DESALEATORIZADO é polinomial. □

É um pouco surpreendente que desaleatorizações produzam, mesmo sem levar em conta como as variáveis aleatórias se distribuem em torno do valor esperado, um algoritmo com razão de aproximação igual à razão esperada do algoritmo probabilístico. No caso do algoritmo acima, o procedimento para o cálculo das esperanças condicionadas é bem simples. Em outros problemas, esses cálculos podem ser bastante complicados, inviabilizando o processo de desaleatorização.

6.3 Geradores de números aleatórios

Uma maneira mais geral de definir algoritmos probabilísticos parte de um gerador de números aleatórios no intervalo $[0, 1)$. Nessa definição alternativa, substitui-se o algoritmo RAND por um algoritmo ideal

RANDUNI RANDUNI que devolve um número real aleatório com distribuição uniforme no intervalo $[0, 1)$ (apêndice D). Existem implementações aproximadas desse algoritmo RAND [Knu98].

Os conceitos introduzidos no início deste capítulo (algoritmo de aproximação probabilístico, algoritmo probabilístico polinomial, etc.) podem ser naturalmente adaptados a esta outra definição. Em particular, os algoritmos vistos neste capítulo são probabilísticos polinomiais também de acordo com esta nova definição, pois é fácil implementar RAND(ρ) com uma chamada a RANDUNI: o algoritmo RAND devolve 1 se o número devolvido por RANDUNI for menor que ρ e devolve 0 em caso contrário.

Por outro lado, não se sabe, em geral, converter um algoritmo que chama RANDUNI um número polinomial de vezes em um que chama RAND um número polinomial de vezes. Por isso é razoável dizer que esta nova definição é uma generalização da original. A desvantagem da nova definição é que ela requer manipulação números reais.

Exercícios

6.1 Considere o seguinte algoritmo para o MAXSAT $(V, \mathcal{C})$.

Algoritmo MAXSAT-CARACOROA (V)
1 $b \leftarrow$ RAND($\frac{1}{2}$)
2 para cada v em V faça $\dot{x}_v \leftarrow b$
3 devolva $\dot{x}$

Mostre que o algoritmo acima é uma 0,5-aproximação probabilística polinomial para o MAXSAT.

6.2 Inspire-se no algoritmo MAXSAT-JOHNSON e projete uma 0,5-aproximação probabilística polinomial para o **problema do corte máximo**:

> **Problema** MAXCUT (G, w): *Dado um grafo G e um peso w_e em $\mathbb{Q}_{\geq}$ para cada aresta e de G, encontrar um corte R que maximize $w(R)$.*

Apresente uma versão desaleatorizada do seu algoritmo.

6.3 Projete uma 0,5-aproximação probabilística polinomial para a seguinte variante do MAXCUT (exercício 6.2), que denotamos por MAXkCUT: dados um grafo G, um peso w_e em $\mathbb{Q}_{\geq}$ para cada aresta e e um inteiro $k \geq 2$, determinar uma partição de V_G em k partes que maximize a soma do peso das arestas com extremos em partes distintas da partição.

6.4 Projete uma 0,5-aproximação probabilística polinomial para o problema da cobertura máxima (exercício 2.4). Mostre que a versão desaleatorizada deste algoritmo é exatamente o algoritmo guloso sugerido no exercício 2.4.

6.5 Escreva uma versão desaleatorizada do algoritmo MAXSAT-GW. Apresente-a da forma mais simples que puder omitindo, se possível, todos os vestígios "probabilísticos" do algoritmo original. Prove, sem usar argumentos probabilísticos, que o algoritmo obtido é uma 0,63-aproximação para MAXSAT.

6.6 Considere o seguinte algoritmo probabilístico para o MAXSAT:

```
Algoritmo MAXSAT-COMBINADO-PROBABILÍSTICO (V, C)
1   b ← RAND (1/2)
2   se b = 0
3       então ẋ ← MAXSAT-JOHNSON (V)
4       senão ẋ ← MAXSAT-GW (V, C)
5   devolva ẋ
```

Prove que esse algoritmo é uma 0,5-aproximação probabilística polinomial para o MAXSAT. Escreva e comente a versão desaleatorizada deste algoritmo. O que muda se alterarmos o $\frac{1}{2}$ da linha 1 do algoritmo para um outro valor no intervalo $(0, 1)$?

6.7 Considere a seguinte versão ponderada do MAXSAT. Além do conjunto V e da coleção $\mathcal{C}$ de cláusulas sobre V, é dado também um peso w_C em $\mathbb{Q}_{\geq}$ para cada cláusula C em $\mathcal{C}$; o problema consiste em encontrar uma valoração que maximize a soma dos pesos das cláusulas satisfeitas. Modifique os três algoritmos apresentados neste capítulo, bem como suas análises, para o MAXSAT ponderado. Que razão de aproximação tem cada um dos algoritmos modificados?

6.8 Considere o seguinte algoritmo para o ESCALONAMENTO (seção 2.1): escolha aleatoriamente a próxima tarefa a ser escalonada e aplique o critério de Graham. Estime o valor esperado da solução assim obtida.

Notas bibliográficas

A discussão do algoritmo MAXSAT-GW é um resumo de um artigo de Goemans e Williamson [GW94]; resumos semelhantes aparecem no livro de Motwani e Raghavan [MR95b] e no livro editado por Hochbaum [Hoc97]. Os exercícios 6.1, 6.3 e 6.4 foram extraídos do livro de Vazirani [Vaz01].

O livro de Motwani e Raghavan [MR95b] é um texto abrangente sobre algoritmos probabilísticos, com muitos exercícios e problemas de pesquisa.

O algoritmo MAXSAT-JOHNSON é atribuído a Johnson [Joh74], embora este tenha escrito a versão não-probabilística do algoritmo. A primeira 0,75-aproximação polinomial para o MAXSAT deve-se a Yannakakis [Yan94]. Zwick [Zwi99] projetou um algoritmo de aproximação para um caso particular do MAXSAT e conjecturou, baseando-se em experimentos númericos, que seu algoritmo é uma 0,797-aproximação para o problema. Recentemente, Asano e Williamson [AW00] projetaram uma 0,784-aproximação, combinando vários algoritmos conhecidos para o problema [GW94, FG95, KZ97]. No momento, esse é o melhor algoritmo de aproximação conhecido para o MAXSAT. Utilizando o algoritmo de Zwick [Zwi99], Asano e Williamson obtêm uma 0,833-aproximação para o MAXSAT, desde que a conjectura de Zwick seja confirmada.

A técnica de arredondamento probabilístico foi introduzida por Raghavan e Thompson [RT87, Rag88] e vem sendo aplicada com bastante sucesso a vários problemas [BTV99, CKR00, Fei99, JV00].

O método das esperanças condicionais aparece implicitamente em um artigo de Erdős e Selfridge [ES73]. A conexão com algoritmos polinomiais determinísticos foi feita por Spencer [Spe87].

Uma outra técnica bem sucedida de desaleatorização de algoritmos probabilísticos foi desenvolvida por Luby [Lub86] e por Alon *et al.* [ABI86]. Uma sucessão de chamadas ao algoritmo RAND produz uma seqüência de *bits* aleatórios. Podemos imaginar então que, além dos dados do problema, um algoritmo probabilístico recebe uma seqüência de *bits* aleatórios, de comprimento conveniente. Freqüentemente, basta que os *bits* nessa seqüência sejam k-mutuamente independentes. Para explicar este conceito, denote por $b_1, \ldots, b_n$ os *bits* produzidos por n chamadas de RAND. Dizemos que $b_1, \ldots, b_n$ são k-**mutuamente independentes** se, para qualquer subseqüência $b_{i_1}, \ldots, b_{i_k}$ de $b_1, \ldots, b_n$, a probabilidade de que $b_{i_1}, \ldots, b_{i_k}$ coincida com qualquer das 2^k seqüências de k *bits* é 2^{-k}. Enquanto independência plena requer um espaço de probabilidade composto de 2^n vetores binários, pode-se construir espaços de probabilidade com $n^{k/2}$ vetores binários cujos *bits* são k-mutuamente independentes. Se k é constante, um algoritmo probabilístico pode ser desaleatorizado da seguinte forma: executa-se o algoritmo para cada um dos vetores neste espaço de probabilidade e devolve-se a melhor das soluções produzidas. Isso resulta em um algoritmo polinomial, determinístico, e com razão de aproximação igual à razão esperada do algoritmo probabilístico original. Para saber mais sobre isso, veja o artigo de Chor e Goldreich [CG89], bem como o relatório técnico de Luby e Wigderson [LW95].

A respeito de geradores de números pseudo-aleatórios, veja os livros de Johnson [Joh87], Devroye [Dev86] e Knuth [Knu98]. Knuth descreve implementações aproximadas do algoritmo RANDUNI que fornecem números pseudo-aleatórios satisfatórios em tempo probabilístico constante: o tempo *esperado* de execução de tais implementações é limitado por uma constante.

Capítulo 7

Programação Semidefinida

Como vimos nos últimos capítulos, relaxações lineares são muito usadas na obtenção de algoritmos de aproximação para problemas de otimização combinatória. A utilização de relaxações lineares deve-se à sua simplicidade e à existência de algoritmos eficientes para resolver programas lineares. Neste capítulo, utilizamos outra classe de relaxações — os **programas semidefinidos** — para a qual também existem bons algoritmos. Vamos ilustrar esse método aplicando-o ao problema do corte máximo.

7.1 Corte máximo

O **problema do corte máximo** (*maximum cut problem*), denotado por MaxCut, é definido como

> **Problema** MaxCut (G, w): *Dados um grafo G e um peso w_e em $\mathbb{Q}_{\geq}$ para cada aresta e, encontrar um corte R que maximize $w(R)$.*

Chamamos o número $w(R)$ de **peso** do corte R. Assim, MaxCut consiste em encontrar um corte de peso máximo.

O problema é NP-difícil, mesmo quando restrito às instâncias em que $w_{ij} = 1$ para toda aresta ij e o grau de cada vértice de G não excede 3 [GJ79].

O problema pode ser formulado da seguinte maneira: dados G e w como acima, encontrar um vetor x indexado por V que

$$\begin{array}{rl} \text{maximize} & \frac{1}{2}\sum_{ij\in E} w_{ij}(1 - x_i x_j) \\ \text{sob as restrições} & x_i x_i = 1 \quad \text{para cada } i \text{ em } V\,, \end{array} \tag{7.1}$$

onde $V := V_G$ e $E := E_G$. Verifiquemos que esse programa é apenas uma reformulação do MAXCUT. Suponha que x é uma solução viável de (7.1), e portanto $x_i \in \{-1, 1\}$ para cada i. Tome $S := \{i : x_i = 1\}$. É claro que $1 - x_i x_j = 2$ se $ij \in \delta(S)$ e $1 - x_i x_j = 0$ em caso contrário. Portanto,

$$\tfrac{1}{2} \textstyle\sum_{ij \in E} w_{ij}(1 - x_i x_j) \;=\; w(\delta(S))\,. \tag{7.2}$$

Suponha agora que S é um conjunto de vértices e defina o vetor x por $x_i := 1$ se $i \in S$ e $x_i := -1$ em caso contrário. Esse vetor é viável em (7.1) e satisfaz a relação (7.2). Diante disso, concluímos que

$$\mathrm{opt}(G, w) \;=\; \tfrac{1}{2} \textstyle\sum_{ij \in E} w_{ij}(1 - x_i^* x_j^*) \tag{7.3}$$

para qualquer solução ótima x^* de (7.1).

O problema (7.1) é um programa quadrático: ele procura um vetor que maximize uma função quadrática sujeita a restrições quadráticas. Obviamente não sabemos resolver este programa quadrático eficientemente, do contrário teríamos um algoritmo exato polinomial para o MAXCUT. Consideremos então uma relaxação de (7.1).

Relaxação vetorial

Uma relaxação vetorial de (7.1) consiste em trocar cada componente de x por um vetor, e portanto trocar x por uma matriz. Mais precisamente, uma relaxação vetorial é o seguinte problema: encontrar uma matriz real Y indexada por[1] $V \times V$ que

$$\begin{array}{rl} \text{maximize} & \tfrac{1}{2} \sum_{ij \in E} w_{ij}(1 - (YY^\top)_{ij}) \\ \text{sob as restrições} & (YY^\top)_{ii} \;=\; 1 \quad \text{para cada } i \text{ em } V\,. \end{array} \tag{7.4}$$

Diz-se que a relaxação (7.4) é vetorial pois suas incógnitas são vetores: as linhas da matriz Y. A relação entre (7.4) e (7.1) é fácil perceber: cada componente x_i de x corresponde à linha $Y_{i\star}$ de Y e cada restrição $x_i x_i = 1$ corresponde à restrição $(YY^\top)_{ii} = 1$, que poderia ser escrita como $Y_{i\star} Y_{i\star} = 1$.

É fácil perceber a relação entre as soluções de (7.4) e (7.1). Considere uma solução viável x de (7.1). Seja k um vértice qualquer e Y

[1] Em princípio, o conjunto de índices de colunas de Y, digamos N, poderia ser diferente de V. Se $|N| = 1$, por exemplo, os programas (7.4) e (7.1) seriam idênticos. Mas, aparentemente [Lov01], o problema (7.4) é NP-difícil se $|N|$ é constante em relação a $|V|$.

a matriz definida por $Y_{\star k} = x$ e $Y_{\star i} = 0$ para cada i diferente de k. É evidente que $(YY^\top)_{ij} = Y_{i\star}Y_{j\star} = x_i x_j$ para cada i e cada j. Em particular, $(YY^\top)_{ii} = x_i x_i = 1$ e portanto Y é viável em (7.4). Ademais, $\sum_{ij} w_{ij}(1 - (YY^\top)_{ij}) = \sum_{ij} w_{ij}(1 - x_i x_j)$. Assim, para qualquer solução ótima $\hat{Y}$ de (7.4) e qualquer solução ótima x^* de (7.1), tem-se $\sum_{ij} w_{ij}(1 - x_i^* x_j^*) \leq \sum_{ij} w_{ij}(1 - (\hat{Y}\hat{Y}^\top)_{ij})$. A relação (7.3) garante então a delimitação superior

$$\mathrm{opt}(G, w) \;\leq\; \tfrac{1}{2} \textstyle\sum_{ij \in E} w_{ij}(1 - (\hat{Y}\hat{Y}^\top)_{ij})\,. \tag{7.5}$$

Resta discutir a existência de uma solução ótima do programa vetorial (7.4). O conjunto das soluções viáveis de (7.4) é um subconjunto fechado e limitado do espaço de todos os vetores reais indexados por $V \times V$. O conjunto é fechado pois é a intersecção dos conjuntos fechados determinados por cada restrição de (7.4). O conjunto é limitado, pois para qualquer Y viável tem-se $\|Y\|^2 = \sum_i Y_{i\star}Y_{i\star} = \sum_i (YY^\top)_{ii} = |V|$. Como $\sum_{ij} w_{ij}(1 - (YY^\top)_{ij})$ é função contínua de Y, o programa (7.4) tem solução ótima.

Algoritmo de Goemans e Williamson

O seguinte algoritmo probabilístico, devido a Goemans e Williamson [GW95b], utiliza uma solução ótima do programa vetorial (7.4) para obter uma solução viável do problema MaxCut.

O algoritmo de Goemans e Williamson depende de um algoritmo que chamamos RandEsfera: ele produz um ponto aleatório com distribuição uniforme na esfera unitária, que é o conjunto de todos os vetores reais r indexados por V tais que $\|r\| = 1$. Por exemplo, cada linha de qualquer matriz Y que satisfaz as restrições de (7.4) é um ponto dessa esfera. RandEsfera

Algoritmo MaxCut-GW (G, w)

1 seja $\hat{Y}$ uma solução ótima de (7.4)
2 $s \leftarrow$ RandEsfera(V)
3 $S \leftarrow \{\, i \in V : s\hat{Y}_{i\star} > 0 \,\}$
4 devolva $\delta(S)$

Antes de calcular uma razão de aproximação do algoritmo, é preciso estabelecer o seguinte lema.

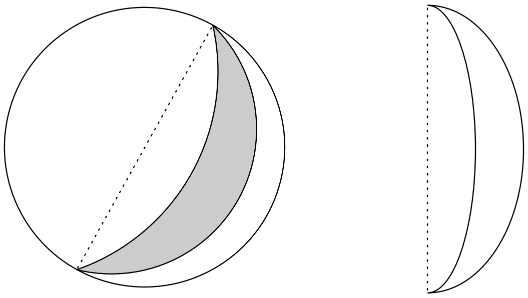


Figura 7.1: Uma "fatia" de uma esfera no $\mathbb{R}^3$.

Lema 7.1: *Para os objetos S e $\hat{Y}$ definidos pelo algoritmo e para quaisquer dois vértices i e j de G,*

$$\mathbf{Pr}[\,ij \in \delta(S)\,] \;\geq\; \frac{1}{\pi} \arccos\left((\hat{Y}\hat{Y}^\top)_{ij}\right).$$

Demonstração: Adote as abreviaturas $\hat{y}$ e $\hat{z}$ para os vetores $\hat{Y}_{i\star}$ e $\hat{Y}_{j\star}$, respectivamente. Claramente,

$$\mathbf{Pr}[\,ij \in \delta(S)\,] \;=\; \mathbf{Pr}[\,s\hat{y} > 0 \text{ e } s\hat{z} \leq 0\,] + \mathbf{Pr}[\,s\hat{y} \leq 0 \text{ e } s\hat{z} > 0\,].$$

O primeiro termo do lado direito é a probabilidade de que s pertença à região da esfera unitária na intersecção dos semi-espaços $\{r : r\hat{y} > 0\}$ e $\{r : r\hat{z} \leq 0\}$. Tal região é a "fatia" da esfera (veja a figura 7.1) de ângulo igual ao ângulo entre os vetores $\hat{y}$ e $\hat{z}$, que denotamos por Θ. Graças à distribuição uniforme (apêndice D), a probabilidade de tal região é proporcional à área da região, e portanto proporcional ao ângulo Θ:

$$\mathbf{Pr}[\,s\hat{y} > 0 \text{ e } s\hat{z} \leq 0\,] \;=\; \frac{\Theta}{2\pi}.$$

De forma semelhante, podemos concluir que $\mathbf{Pr}[\,s\hat{y} \leq 0 \text{ e } s\hat{z} > 0\,] = \Theta/2\pi$, terminando assim a prova do lema, já que $\Theta = \arccos(\hat{y}\,\hat{z})$. $\Box$

Alguns cálculos elementares mostram que (veja figura 7.2), para todo x no intervalo $[-1, 1]$,

$$\frac{1}{\pi} \arccos(x) \;\geq\; 0{,}878\,\frac{1}{2}(1 - x). \tag{7.6}$$

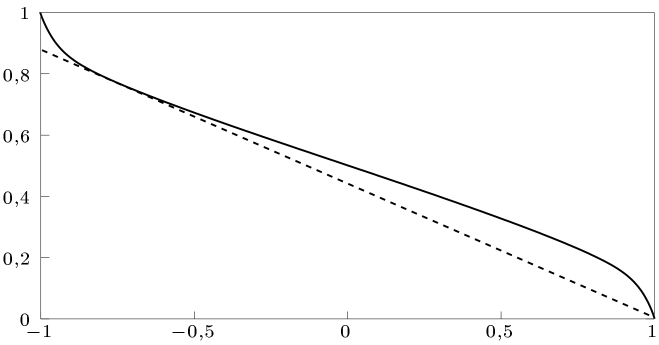


Figura 7.2: Gráfico das funções $f(x) := \frac{1}{\pi}\arccos(x)$ e $g(x) := 0{,}878\,\frac{1}{2}(1-x)$. Verifica-se que $f(x) \geq g(x)$ no intervalo $[-1,1]$.

Teorema 7.2: *Se X é a variável aleatória*[2] *cujo valor é o peso de um corte produzido pelo pelo algoritmo* MaxCut-GW (G,w) *então*

$$\mathbf{E}[X] \;\geq\; 0{,}878\,\mathrm{opt}(G,w)\;.$$

Demonstração: Em virtude do lema 7.1 e de (7.6), temos que

$$\begin{aligned} \mathbf{E}[X] &= \textstyle\sum_{ij\in E} w_{ij}\,\mathbf{Pr}[\,ij \in \delta(S)\,] \\ &\geq \textstyle\sum_{ij\in E} w_{ij}\,\dfrac{1}{\pi}\arccos\big((\hat{Y}\hat{Y}^{\top})_{ij}\big) \\ &\geq 0{,}878\,\dfrac{1}{2}\textstyle\sum_{ij\in E} w_{ij}(1-(\hat{Y}\hat{Y}^{\top})_{ij}) \\ &\geq 0{,}878\,\mathrm{opt}(G,w)\;, \end{aligned} \tag{7.7}$$

onde (7.7) segue de (7.5). □

O algoritmo MaxCut-GW supõe um modelo de computação (apêndice E) muito mais poderoso e menos realista que o dos capítulos anteriores: o modelo é capaz de manipular números reais arbitrários e supõe que cada operação aritmética consome uma quantidade de tempo que não depende dos valores dos operandos. Veremos na próxima seção que a linha 1 do algoritmo MaxCut-GW pode ser executada em tempo polinomial em $\langle G\rangle + \langle w\rangle$ no modelo de computação que estamos adotando. A linha 2 envolve uma execução do algoritmo RandEsfera,

[2] O espaço de probabilidades é definido pela distribuição uniforme sobre a esfera unitária (exemplo D.3).

que é probabilístico polinomial (seção 6.3), pois pode ser implementado com $O(|V|)$ chamadas à função RANDUNI. A linha 3 consome tempo $O(|V|^2)$. Assim, o algoritmo é probabilístico polinomial.

Teorema 7.3: *O algoritmo* MAXCUT-GW *é uma* 0,878-*aproximação probabilística polinomial para o* MAXCUT. □

7.2 Programas semidefinidos

Um **programa vetorial** consiste no seguinte: dada uma matriz W indexada por $V \times V$, uma matriz A indexada por $M \times (V \times V)$ e um vetor b indexado por M, encontrar uma matriz Y indexada por $V \times V$ que

$$\begin{array}{rll} \text{minimize} & \sum_{ij \in V \times V} W_{ij} (YY^\top)_{ij} & \\ \text{sob as restrições} & \sum_{ij \in V \times V} A_{kij} (YY^\top)_{ij} \ = \ b_k & \text{para cada } k \text{ em } M\,. \end{array}$$

PV( ) Vamos denotar esse programa vetorial por $\mathrm{PV}(W, A, b)$. É fácil perceber que o problema (7.4) é uma instância de PV.

Como mostraremos a seguir, qualquer programa PV pode ser resolvido[3] em tempo polinomial, no modelo de computação que estamos adotando. O algoritmo dos elipsóides (mencionado no apêndice C em conexão com programas lineares) pode ser usado para resolver programas não-lineares cujo conjunto das soluções viáveis seja fechado e convexo[4] [GLS93, Sch86]. O conjunto das soluções viáveis do programa vetorial $\mathrm{PV}(W, A, b)$ não é convexo, mas o programa pode ser convertido, como veremos em seguida, em um programa semidefinido equivalente cujo conjunto das soluções viáveis é convexo.

Dizemos que uma matriz quadrada X é **positiva semidefinida**, o
$X \succeq 0$ que denotamos por $X \succeq 0$, se $X = YY^\top$ para alguma matriz quadrada real Y. Um **programa semidefinido** consiste no seguinte: dada uma matriz W indexada por $V \times V$, uma matriz A indexada por $M \times (V \times V)$ e um vetor b indexado por M, encontrar uma matriz X indexada por

[3] Mais precisamente, é possível encontrar uma solução "aproximada" com qualquer precisão prefixada.

[4] Um conjunto $\mathcal{Y}$ de vetores é **convexo** se, para quaisquer y e y' em $\mathcal{Y}$ e qualquer λ no intervalo fechado $[0,1]$, o vetor $\lambda y + (1-\lambda)y'$ está em $\mathcal{Y}$. O conceito de conjunto convexo de matrizes é análogo.

$V \times V$ que

$$\begin{array}{rrcll} \text{minimize} & \sum_{ij} W_{ij} X_{ij} & & & \\ \text{sob as restrições} & \sum_{ij} A_{kij} X_{ij} & = & b_k & \text{para cada } k \text{ em } M \text{ e} \\ & X & \succeq & 0 & . \end{array}$$

Denotamos esse problema por $\mathrm{PSD}(W, A, b)$. Se o conjunto das soluções viáveis de $\mathrm{PSD}(W, A, b)$ for limitado, o problema tem solução ótima.[5] PSD( )

Há uma correspondência simples entre soluções viáveis do $\mathrm{PV}(W, A, b)$ e soluções viáveis do $\mathrm{PSD}(W, A, b)$. Se X é uma solução viável do $\mathrm{PSD}(W, A, b)$, então a equação $YY^\top = X$ define uma matriz Y que é solução viável do $\mathrm{PV}(W, A, b)$; e reciprocamente. Em particular, se X é uma solução ótima do $\mathrm{PSD}(W, A, b)$, então Y é uma solução ótima do $\mathrm{PV}(W, A, b)$.

Existem algoritmos elementares (apêndice B) que determinam, para qualquer matriz positiva semidefinida X, uma matriz real Y tal que $YY^\top = X$. Assim, é possível resolver um programa vetorial da seguinte maneira: obtenha uma solução ótima X do programa semidefinido; calcule uma matriz Y tal que $YY^\top = X$; essa Y é uma solução ótima do problema vetorial.

O conjunto das soluções viáveis de qualquer programa semidefinido é convexo e fechado. Ademais, o programa semidefinido derivado do MAX-CUT (bem como os programas derivados de muitos outros problemas de otimização combinatória) é limitado e assim pode ser resolvido em tempo polinomial por meio do algoritmo dos elipsóides, desde que tenhamos um algoritmo de separação apropriado. Um tal algoritmo de separação é parte do algoritmo de decomposição de matrizes positivas semidefinidas a que aludimos acima: se a matriz simétrica X dada não é positiva semidefinida, o algoritmo exibe uma violação da restrição $X \succeq 0$, ou seja, um vetor real y para o qual $yXy < 0$.

7.3 Considerações práticas

Neste capítulo usamos, até agora, um modelo de computação pouco realista. O algoritmo MAXCUT-GW pode ser convertido, de maneira óbvia, em um algoritmo "aproximado" para um modelo de computação

[5] Via de regra, as soluções ótimas não são racionais, mesmo que W, A e b sejam racionais.

mais realista: basta executar todos os cálculos com alguma precisão finita. (Determinar uma precisão dos cálculos que produza razão de aproximação próxima de 0,878 não é tarefa fácil.) O algoritmo RANDESFERA pode também ser implementado [Joh87, Dev86, Knu98] de maneira aproximada a partir de um gerador de números aleatórios com distribuição uniforme no intervalo $[0, 1)$ (exercícios D.3 e D.4, apêndice D).

O algoritmo dos elipsóides, bem como qualquer outro algoritmo usado para resolver programas semidefinidos, fornece apenas soluções aproximadas de tais programas. Isso é inevitável, já que as soluções ótimas de $\mathrm{PSD}(W, A, b)$ podem ter componentes irracionais [Goe97]. O algoritmo dos elipsóides pode ser usado para produzir [GLS93], para qualquer ε positivo, uma matriz X que satisfaz, a menos de ε, as restrições de $\mathrm{PSD}(W, A, b)$.

Exercícios

7.1 Verifique que o conjunto das soluções viáveis do programa vetorial (7.4) não é, em geral, convexo.

7.2 Escreva o programa semidefinido correspondente a (7.4). Verifique que o conjunto das soluções viáveis deste programa semidefinido é convexo.

7.3 Verifique que o conjunto das soluções viáveis do programa semidefinido correspondente a (7.4) é fechado e limitado.

7.4 Descreva um programa vetorial cuja solução ótima tem valor irracional.

7.5 Prove a desigualdade (7.6).

7.6 A restrição do MAXSAT (capítulo 6) a instâncias $(V, \mathcal{C})$ em que cada cláusula em $\mathcal{C}$ tem exatamente duas variáveis é conhecida como MAX2SAT. Formule o MAX2SAT como um programa quadrático.

7.7 Formule o MAXkCUT (exercício 6.3) como um programa quadrático. Exiba uma relaxação vetorial deste programa e uma relaxação semidefinida dele.

7.8 Considere o problema MAXCUT com a seguinte restrição adicional: são dados dois conjuntos de vértices S_1 e S_2 e queremos um corte de peso máximo que deixe os vértices de S_1 em um mesmo componente e separe quaisquer dois vértices em S_2. Formule esse problema como um programa quadrático e exiba uma relaxação vetorial dele. Mostre que o algoritmo MaxCut-GW pode ser adaptado para esse problema, conservando a mesma razão de aproximação esperada.

Notas bibliográficas

A apresentação desse capítulo inspirou-se no artigo de Goemans e Williamson [GW95b], nas notas de aula de Williamson [Wil98], no *Handbook of Combinatorics* [GGL95] e no texto de Lovász [Lov01]. Os exercícios 7.6, 7.7 e 7.8 foram extraídos do livro do Vazirani [Vaz01].

Programação semidefinida e suas aplicações à otimização combinatória são discutidas por Deza e Laurent [DL97] e por Lovász [Lov01].

Antes do aparecimento do algoritmo MaxCut-GW, a melhor razão de aproximação conhecida para o problema MaxCut era 0,5 (algoritmo de Erdős [Erd67], exercício 6.2). Karloff [Kar96] apresentou uma família de instâncias (G, w) do MaxCut para as quais o algoritmo MaxCut-GW produz cortes de peso arbitrariamente próximo de $0{,}878\,\mathrm{opt}(G, w)$. Mahajan e Ramesh [MR95a] mostraram como desaleatorizar o algoritmo MaxCut-GW. A desaleatorização é bem mais complexa que aquela apresentada no capítulo 6.

Goemans e Williamson aplicaram o método de programação semidefinida a vários problemas [GW95b]. Eles obtiveram uma 0,758-aproximação para o MaxSat, uma 0,878-aproximação para o Max2Sat, definido no exercício 7.6, e uma 0,796-aproximação para a versão do MaxCut em grafos orientados, conhecida por MaxDiCut.

Feige e Goemans [FG95] obtiverem uma 0,931-aproximação para o Max2Sat e uma 0,859-aproximação para o MaxDiCut. Karloff e Zwick [KZ97] propuseram uma 0,875-aproximação para o problema Max3Sat, análogo ao Max2Sat.

Skutella [Sku98] aplicou programação semidefinida a uma variante do problema Escalonamento, discutido na seção 2.1. Karger, Motwani e Sudan [KMS98] apresentaram uma $(n^{1-3/(k+1)} \log^{1/2} n)$-aproximação para o problema de coloração de vértices em grafos com n vértices e número cromático é no máximo k. Vários outros trabalhos recentes [AK98, BKRV98, CS98, CRW01, FJ97, GW01, Hal00, KG98, Sku99, Zwi99] também usam programação semidefinida.

De acordo com Alizadeh [Ali95], métodos de pontos interiores podem ser mais eficientes que o algoritmo dos elipsóides na resolução de programas semidefinidos. Mais informações sobre programação semidefinida podem ser encontradas no livro de Wolkowicz, Saigal e Vandenberghe [WSV00].

Capítulo 8

Inaproximabilidade

Por que alguns problemas de otimização parecem ser mais difíceis de aproximar do que outros? Como podemos medir ou comprovar estes diferentes graus de dificuldade? O presente capítulo trata de questões desse tipo.

Cook [Coo71] mostrou que a classe NP contém problemas completos, ou seja, problemas que são pelo menos tão difíceis quanto qualquer outro problema na classe. Estes são os chamados problemas NP-completos (apêndice E). Logo após, Karp [Kar72] apresentou uma grande coleção de problemas NP-completos, que incluía vários problemas de decisão correspondentes a problemas de otimização combinatória bem conhecidos. É crença geral que não existem algoritmos polinomiais para resolver problemas NP-completos, ou seja, acredita-se que $\mathrm{P} \neq \mathrm{NP}$.

Assim como problemas de decisão são classificados conforme a sua dificuldade computacional, problemas de otimização são classificados conforme o seu grau de aproximabilidade por algoritmos polinomiais. Neste capítulo são definidas as classes PO, FPTAS, PTAS, APX e NPO de problemas de otimização.

Nos capítulos anteriores foram estudados diversos algoritmos de aproximação para vários problemas NP-difíceis. Para nenhum dos problemas estudados foi apresentada evidência de que não existem algoritmos com melhor razão de aproximação. Neste capítulo mostramos que, freqüentemente, a dificuldade em se obter uma melhor razão de aproximação está intrinsecamente ligada à questão "$\mathrm{P} \neq \mathrm{NP}$?". Apresentamos resultados de inaproximabilidade que têm tipicamente a seguinte forma:

se existe uma α-aproximação polinomial para Π, então $\mathrm{P} = \mathrm{NP}$,

onde Π é um problema de otimização. Um tal resultado indica que a existência de uma α-aproximação polinomial para Π é improvável. Por outro lado, apesar de não se acreditar que este seja o caso, se P = NP então, em particular, existe um algoritmo exato polinomial para cada problema tratado no presente texto e toda a discussão sobre algoritmos de aproximação perde parte de seu sentido.

8.1 Classes de problemas de otimização

Um **problema de otimização** tem três ingredientes, a saber, um
Sol(I) conjunto de instâncias, um conjunto Sol(I) de soluções viáveis de cada instância I e uma função val que associa a cada instância I e solução
val(I, S) S em Sol(I) um valor racional não-negativo val(I, S). Se o problema é **de minimização** o objetivo é encontrar, para qualquer instância dada, uma solução viável de valor mínimo. Se o problema é **de maximização** o objetivo é encontrar uma solução viável de valor máximo. Em suma, um problema de otimização Π consiste no seguinte:

> **Problema** $\Pi\,(I)$: *Dada uma instância I, encontrar S em* Sol(I) *que minimize (maximize)* val(I, S).

Uma **solução ótima** para um problema de otimização é uma solução viável que maximize ou minimize, dependendo do caso, o valor da função val. Denotamos por opt(I) o valor de uma solução ótima da instância I.

NPO A classe de problemas NPO, que é a extensão de NP a problemas de otimização, é formada pelos problemas de otimização para os quais:

- existe uma função polinomial p tal que $\langle S\rangle \leq p(\langle I\rangle)$ para toda instância I do problema e toda solução viável S de I;
- existe um algoritmo polinomial que decide se uma dada palavra é uma representação válida de uma instância do problema;
- existe um algoritmo polinomial que decide se um dado objeto é solução viável de uma dada instância do problema;
- existe um algoritmo polinomial que calcula val(I, S), dados I e S.

PO Denotamos pela sigla PO o conjunto dos problemas em NPO para os quais existe um algoritmo exato polinomial. Esta é, portanto, uma
APX extensão da classe P a problemas de otimização. A classe APX é formada pelos problemas em NPO para os quais existe uma α-aproximação polinomial para alguma constante α.

Um **esquema de aproximação** (*approximation scheme*) para um problema de otimização é um algoritmo A que recebe um número racional positivo ε e uma instância I e devolve uma solução viável $A(\varepsilon, I)$ com um erro relativo de no máximo ε, ou seja,

$$(1-\varepsilon)\,\mathrm{opt}(I) \leq \mathrm{val}(I, A(\varepsilon, I)) \leq (1+\varepsilon)\,\mathrm{opt}(I)\ .$$

No caso de problema de maximização, somente a desigualdade esquerda é relevante; ela só faz sentido para ε no intervalo $(0,1)$. Se o problema é de minimização, somente a desigualdade direita interessa.[1] Dizemos que A é um **esquema de aproximação polinomial** (*polynomial-time approximation scheme*) se o algoritmo $A(\varepsilon, \cdot)$ é polinomial para todo ε fixo. Ademais, quando a quantidade de tempo consumida pelo algoritmo A é limitada por $p(\langle I\rangle, 1/\varepsilon)$ para alguma função polinomial p, então A é esquema de aproximação **plenamente polinomial** (*fully polynomial-time approximation scheme*). Para o problema da mochila (MOCHILA), tratado no capítulo 2, o algoritmo MOCHILA-IK$_\varepsilon$ é um esquema de aproximação plenamente polinomial.[2]

A classe PTAS é composta pelos problemas em NPO que possuem um esquema de aproximação polinomial e a classe FPTAS, pelos que admitem um esquema de aproximação plenamente polinomial. Das definições, segue imediatamente que PTAS FPTAS

$$\mathrm{PO} \subseteq \mathrm{FPTAS} \subseteq \mathrm{PTAS} \subseteq \mathrm{APX} \subseteq \mathrm{NPO}\ .$$

Como será observado na próxima seção, se alguma das inclusões acima não for estrita, então P = NP. Por outro lado, se P = NP, então PO = NPO.

Como pode ser verificado, todos os problemas de otimização estudados ao longo do presente texto estão em NPO. Assim, os algoritmos de aproximação vistos distribuem esses problemas nas classes definidas acima. Com exceção do teorema 8.2, todos os teoremas a seguir estão, essencialmente, demonstrados nos capítulos anteriores.

Teorema 8.1: *O problema* MOCHILA *está em* FPTAS. □

Teorema 8.2 (Hochbaum e Shmoys [HS88]): *O problema* ESCALONAMENTO *está em* PTAS. □

---

[1] Se o problema é de maximização e ε está fixo, temos uma $(1-\varepsilon)$-aproximação. Se o problema é de minimização, temos uma $(1+\varepsilon)$-aproximação.

[2] A quantidade de tempo consumida pelo algoritmo MOCHILA-IK$_\varepsilon$ é proporcional a n^3/ε, onde n é o número de objetos.

Teorema 8.3: *Os problemas* EMPACOTAMENTO, MAXCUT, MAXSAT, MINCV, MINFS *e* TSPM *estão em* APX. □

Teorema 8.4: *Os problemas* MINCC, MINMCUT, MINTC *e* TSP *estão em* NPO. □

8.2 NP-completude e inaproximabilidade

Nesta seção, mostramos que a dificuldade em obter-se uma melhor razão de aproximação para alguns problemas tratados ao longo do texto é herdada da teoria de NP-completude e da clássica questão "P $\neq$ NP?".

Teorema 8.5: *Se* APX = NPO, *então* P = NP.

Demonstração: Pelo teorema 8.4, o problema TSP está em NPO. Logo, é suficiente mostrar o seguinte teorema de Sahni e Gonzalez [SG76]: se o problema TSP está em APX, então P = NP.

Seja A uma α-aproximação polinomial para o TSP, onde α é uma constante. Utilizando A, pode-se criar um algoritmo polinomial que resolve o problema do circuito hamiltoniano, denotado por PCH. O algoritmo é o seguinte. Dado um grafo G, instância arbitrária do PCH, construa a instância (K, c) do TSP, onde K é o grafo completo com conjunto V_G de vértices e custo $c_e = 1$ se a aresta e está em G e $c_e = \alpha\,|V_G|$ se e não está em G. Uma solução viável da instância (K, c) do TSP é um circuito hamiltoniano em K e o valor da solução viável é o custo do circuito. Aplique o algoritmo A à instância (K, c). A execução do algoritmo consome uma quantidade de tempo limitada por uma função polinomial de $\langle G \rangle$.

Se G é hamiltoniano então $\mathrm{opt}(K, c) = |V_G|$ e, se G não é hamiltoniano, então $\mathrm{opt}(K, c) > \alpha\,|V_G|$. Donde, $\mathrm{val}((K, c), A(K, c)) \leq \alpha\,|V_G|$ se e somente se G é hamiltoniano. A existência de um tal algoritmo polinomial mostra que PCH está em P. Como PCH é NP-completo [Kar72], temos que P = NP. □

Teorema 8.6: *Se* PTAS = APX, *então* P = NP.

Demonstração: Pelo teorema 8.3, o problema EMPACOTAMENTO está em APX. Portanto, basta provar que se o problema EMPACOTAMENTO está em PTAS, então P = NP.

Seja A um esquema de aproximação polinomial para o problema

EMPACOTAMENTO. Utilizando o algoritmo[3] $A(1/3, \cdot)$, pode-se projetar um algoritmo polinomial o problema da partição (*partition*).

> **Problema** PARTIÇÃO (n, v): *Dados um inteiro positivo n e, para cada i em $\{1, \ldots, n\}$, um número v_i em $\mathbb{Q}_{\geq}$, decidir se existe uma partição $\{B_1, B_2\}$ de $\{1, \ldots, n\}$ tal que $v(B_1) = v(B_2)$.*

O algoritmo polinomial para PARTIÇÃO é muito simples. Dada uma instância arbitrária (n, v) do problema, calcule $\sigma := (\sum_i v_i)/2$. Podemos supor que $\sigma \neq 0$ e que $v_i \leq \sigma$ para todo i. Seja $c_i := v_i/\sigma$ para cada i. Utilize o algoritmo $A(1/3, \cdot)$ para resolver EMPACOTAMENTO(n, c). Isso tudo consome uma quantidade de tempo limitada por uma função polinomial de $\langle n \rangle + \langle v \rangle$.

Se a resposta a PARTIÇÃO(n, v) é SIM então o valor ótimo $\mathrm{opt}(n, c)$ de EMPACOTAMENTO(n, c) é 2; caso contrário é pelo menos 3. Como $\mathrm{val}((n, c), A(1/3, (n, c)))$ é um número inteiro e

$$\mathrm{val}((n, c), A(1/3, (n, c))) \leq (1 + 1/3)\,\mathrm{opt}(n, c) = (4/3)\,\mathrm{opt}(n, c)\ ,$$

a resposta do problema PARTIÇÃO(n, v) é SIM se e somente se $\mathrm{val}((n, c), A(1/3, (n, c))) = 2$. Logo, o algoritmo proposto resolve PARTIÇÃO em tempo polinomial. Como PARTIÇÃO é NP-completo [Kar72], temos que P = NP. □

Se I é uma instância de um problema, então $\mathrm{Max}(I)$ é definido como o maior número inteiro em valor absoluto que ocorre em I; defina $\mathrm{Max}(I) := 0$ se nenhum número inteiro ocorre em I (MAXSAT é um exemplo em que isso ocorre). Estamos supondo que os números racionais são representados como quociente entre dois números inteiros. Assim, por exemplo, se o número $3/17$ ocorre em I temos que $\mathrm{Max}(I) \geq 17$. $\mathrm{Max}(I)$

> **Teorema 8.7** (Garey e Johnson [GJ78]): *Seja Π um problema de otimização NP-difícil no sentido forte tal que $\mathrm{val}(I, S)$ é um número inteiro não-negativo para toda instância I e todo S em $\mathrm{Sol}(I)$. Seja ainda $p(n, m)$ uma função polinomial tal que*
>
> $$\mathrm{opt}(I) \leq p(\langle I \rangle, \mathrm{Max}(I))$$
>
> *para toda instância I de Π. Se Π está em FPTAS, então P = NP.*

[3] Na prova, $1/3$ pode ser substituído por qualquer racional no intervalo $(0, 1/2)$.

Demonstração: Suponha que Π é um problema de minimização (argumentos simétricos se aplicam a problemas de maximização) e seja A um esquema de aproximação plenamente polinomial para Π. Utilizando o esquema A, pode-se projetar um algoritmo pseudopolinomial para resolver o problema Π, provando assim que $\mathrm{P} = \mathrm{NP}$.

Dada uma instância I, compute $\varepsilon := 1/(p(\langle I\rangle, \mathrm{Max}(I))+1)$ e aplique o algoritmo A aos argumentos ε e I. Em uma quantidade de tempo limitada por uma função polinomial em $\langle I\rangle$ e $1/\varepsilon$, e portanto em tempo pseudopolinomial, o algoritmo A devolve um elemento $A(\varepsilon, I)$ em $\mathrm{Sol}(I)$ que satisfaz $\mathrm{val}(I, A(\varepsilon, I)) \leq (1+\varepsilon)\,\mathrm{opt}(I)$, ou seja,

$$\begin{aligned} \mathrm{val}(I, A(\varepsilon, I)) - \mathrm{opt}(I) &\leq \varepsilon\,\mathrm{opt}(I) \\ &= \frac{\mathrm{opt}(I)}{p(\langle I\rangle, \mathrm{Max}(I)) + 1} \\ &< 1\,, \end{aligned}$$

para toda instância I. Como $\mathrm{val}(I, A(\varepsilon, I))$ e $\mathrm{opt}(I)$ são inteiros, $\mathrm{val}(I, A(\varepsilon, I)) = \mathrm{opt}(I)$. Ou seja, o algoritmo pseudopolinomial projetado resolve Π exatamente. Como Π é NP-difícil no sentido forte, temos que $\mathrm{P} = \mathrm{NP}$. □

No teorema 8.7, a hipótese dos valores das soluções viáveis serem números inteiros é apenas aparentemente restritiva. Todos os problemas formulados ao longo do texto envolvem números racionais. Entretanto, cada um desses problemas pode ser reformulado como um problema em que os valores das soluções viáveis são números inteiros e que é polinomialmente equivalente ao problema original. Mais ainda, um algoritmo de aproximação para um dos problemas pode ser transformado em um algoritmo de mesma razão de aproximação para o outro. Considere, por exemplo, a reformulação abaixo do problema ESCALONAMENTO.

> **Problema** ESCALONAMENTOINT (m, n, t): *Dados inteiros positivos m e n e um tempo t_i em $\mathbb{Z}_{\geq}$ para cada i em $\{1, \ldots, n\}$, encontrar uma partição $\{M_1, \ldots, M_m\}$ de $\{1, \ldots, n\}$ que minimize* $\max_j t(M_j)$.

Os problemas ESCALONAMENTO e ESCALONAMENTOINT são polinomialmente equivalentes. Ademais, uma α-aproximação polinomial para um dos problemas pode ser transformada em uma α-aproximação polinomial para o outro. Logo, pelo teorema 8.2, ESCALONAMENTOINT também está em PTAS.

As hipóteses principais do teorema 8.7 são que as soluções viáveis de Π têm valor inteiro "não muito grande" e que Π é NP-difícil no sentido forte. Estas hipóteses se aplicam a vários problemas vistos neste texto, tais como, MaxSat, MaxCut, MinCV e o recém definido EscalonamentoInt, que estão em APX. Desta forma, se algum desses problemas admitir um esquema de aproximação plenamente polinomial, então P = NP. Como, em particular, EscalonamentoInt está em PTAS, então é improvável que existam esquemas de aproximação plenamente polinomiais para todos os problemas em PTAS.

Teorema 8.8: *Se* FPTAS = PTAS, *então* P = NP. □

Como Mochila é um problema NP-difícil que está em FPTAS, então é improvável que existam algoritmos polinomiais para todos os problemas em FPTAS.

Teorema 8.9: *Se* PO = FPTAS, *então* P = NP. □

Finalmente, o teorema a seguir mostra que, com exceção do teorema 8.7, vale a recíproca de cada teorema visto nesta seção.

Teorema 8.10: *Se* P = NP, *então* PO = NPO. □

Uma prova do teorema 8.10 pode ser encontrada em Ausiello, Crescenzi, Gambosi, Kann, Marchetti-Spaccamela e Protasi [ACG$^+$99].

8.3 Completude para problemas de otimização

Informalmente, problemas completos de uma certa classe são aqueles pelo menos tão difíceis quanto qualquer outro da classe. Para a classe NP isto significa que um algoritmo polinomial para um problema NP-completo pode ser transformado em um algoritmo polinomial para resolver qualquer outro problema em NP. Portanto, se algum problema NP-completo está em P, então P = NP. O conceito de completude pode ser estendido para problemas de otimização. Da mesma maneira que foi conveniente para a classe NP utilizar uma redução que preserva polinomialidade, para problemas de otimização é necessário um tipo de redução que, além da polinomialidade, preserve também a razão de aproximação. Concretamente, usaremos uma redução que preserva a existência de um esquema de aproximação polinomial.

Uma **AP-redução** de um problema de otimização Π a um problema de otimização Π' é um terno (f, g, β) em que f e g são algoritmos e β é um número racional positivo tais que:

(AP1) f recebe um número racional positivo δ e uma instância I de Π, e devolve uma instância $f(\delta, I)$ de Π';

(AP2) g recebe um número racional positivo δ, uma instância I de Π e um elemento S' em $\mathrm{Sol}(f(\delta, I))$, e devolve $g(\delta, I, S')$ em $\mathrm{Sol}(I)$;

(AP3) para todo número racional positivo δ, os algoritmos $f(\delta, \cdot)$ e $g(\delta, \cdot, \cdot)$ são polinomiais; e

(AP4) para toda instância I de Π, todo número racional positivo δ, e todo S' em $\mathrm{Sol}(f(\delta, I))$, vale que se

$$(1-\delta)\,\mathrm{opt}(f(\delta, I)) \leq \mathrm{val}(f(\delta, I), S') \leq (1+\delta)\,\mathrm{opt}(f(\delta, I))\ ,$$

$$\text{então } (1-\beta\delta)\,\mathrm{opt}(I) \leq \mathrm{val}(I, g(\delta, I, S')) \leq (1+\beta\delta)\,\mathrm{opt}(I)\ .$$

$\leq_{\mathrm{AP}}$ Será usada a notação $\Pi \leq_{\mathrm{AP}} \Pi'$ para denotar a existência de uma AP-redução de Π a Π'. Dizemos que Π pode ser AP-reduzido a Π' significando que $\Pi \leq_{\mathrm{AP}} \Pi'$.

Teorema 8.11: *Se* $\Pi_1 \leq_{\mathrm{AP}} \Pi_2$ *e* $\Pi_2 \leq_{\mathrm{AP}} \Pi_3$, *então* $\Pi_1 \leq_{\mathrm{AP}} \Pi_3$. □

Teorema 8.12: *Se* Π *está em* NPO, Π' *está em* APX *e* $\Pi \leq_{\mathrm{AP}} \Pi'$, *então* Π *está em* APX. □

O próximo teorema mostra que AP-redução preserva a existência de um esquema de aproximação polinomial.

Teorema 8.13: *Se* Π *está em* NPO, Π' *está em* PTAS *e* $\Pi \leq_{\mathrm{AP}} \Pi'$, *então* Π *está em* PTAS.

Demonstração: Seja A' um esquema de aproximação polinomial para Π' e seja (f, g, β) uma AP-redução de Π a Π'. Utilizando o esquema A' e os algoritmos f e g será criado um esquema de aproximação polinomial A para Π. O algoritmo A abaixo recebe um número racional positivo ε e uma instância I de Π e devolve S em $\mathrm{Sol}(I)$ com erro relativo de no máximo ε.

Algoritmo $A(\varepsilon, I)$

$\delta \leftarrow \varepsilon/\beta$
$I' \leftarrow f(\delta, I)$
$S' \leftarrow A'(\delta, I')$
$S \leftarrow g(\delta, I, S')$
devolva S

$$\begin{tikzcd} I \arrow[r, "f(\delta,I)"] \arrow[d, "A(\varepsilon,I)"'] & I' \arrow[d, "A'(\delta,I')"] \\ S & S' \arrow[l, "g(\delta,I,S')"] \end{tikzcd}$$

De (AP3) e do fato de A' ser um esquema de aproximação polinomial, segue que, para todo ε, o algoritmo $A(\varepsilon,\cdot)$ é polinomial. As propriedades (AP1) e (AP2) e a definição de esquema de aproximação garantem que I' é uma instância de Π', que S' está em $\mathrm{Sol}(I') = \mathrm{Sol}(f(\delta, I))$, que S está em $\mathrm{Sol}(I)$, e que

$$(1-\delta)\,\mathrm{opt}(I') \le \mathrm{val}(I', S') = \mathrm{val}(f(\delta, I), S') \le (1+\delta)\,\mathrm{opt}(f(\delta, I))\,.$$

Logo, de (AP4), tem-se que

$$(1-\varepsilon)\,\mathrm{opt}(I) \le \mathrm{val}(I, S) \le (1+\varepsilon)\,\mathrm{opt}(I)\,,$$

já que $\beta\,\delta = \varepsilon$. Em suma, $A(\varepsilon,\cdot)$ é um algoritmo polinomial que recebe uma instância I de Π e devolve uma solução viável $A(\varepsilon, I)$ com erro relativo de no máximo ε, como desejado. Como Π está em NPO, tem-se que Π está em PTAS. □

Um problema Π em APX é **APX-completo** se cada problema em APX pode ser AP-reduzido a Π. O primeiro problema que se demonstrou ser APX-completo foi MaxSat.

Teorema 8.14 (Papadimitriou e Yannakakis [PY91], Khanna, Motwani, Sudan e Vazirani [KMSV99]): *O problema* MaxSat *é* APX-*completo.* □

Um problema Π, não necessariamente em APX, é **APX-difícil** se a existência de um esquema de aproximação polinomial para Π implica em P = NP. Assim, por exemplo, MaxSat é APX-difícil, já que, pelos teorema 8.6 e 8.13, a existência de um esquema de aproximação polinomial para qualquer problema APX-completo implica em P = NP. Note que, se PTAS $\neq$ APX, então todo problema APX-completo está em APX $\setminus$ PTAS.

Teorema 8.15: *Os problemas* Empacotamento, MaxCut, MaxSat, MinCC, MinCV, MinFS, MinMCut, MinTC, TSPM *e* TSP *são* APX-*difíceis.* □

Um problema Π em NPO é **NPO-completo** se cada problema em NPO pode ser AP-reduzido a Π. Observe que se APX $\neq$ NPO, então todo problema NPO-completo está em NPO $\setminus$ APX.

Teorema 8.16 (Orponen e Mannila [OM87]): *O problema* TSP *é* NPO*-completo.* $\square$

8.4 Limiares de aproximação

O **limiar de aproximação** (*approximation threshold*) de um problema de minimização é o maior limitante inferior de todos os α para os quais existe uma α-aproximação polinomial para o problema. No caso de problema de maximização troca-se, na definição acima, maior limitante inferior por menor limitante superior.

É claro que se P = NP então o limiar de aproximação de todo problema em NPO é 1. Na tabela 8.1 são apresentadas, sob a hipótese de P $\neq$ NP, delimitações para os limiares de aproximação de problemas abordados nesse texto.

problema	limiar de aproximação
MOCHILA	$= 1$ [IK75] (o problema está em FPTAS)
ESCALONAMENTO	$= 1$ [HS88] (o problema está em PTAS)
EMPACOTAMENTO	$\geq 3/2$ [GJ79]
MAXCUT	$\leq 16/17$ [Hås97]
MAXSAT	$\leq 7/8$ [Hås97]
MINCV	$\geq 7/6$ [Hås97]
TSPM	$\geq 131/130$ [EK00]
MINCC $(E, \mathcal{S}, c)$	$> \varepsilon \log \lvert E \rvert$, para alguma constante $\varepsilon > 0$ [RS97]
MINTC $(E, \mathcal{S}, c)$	$> \varepsilon \log \lvert E \rvert$, para alguma constante $\varepsilon > 0$ (equivalente ao MINCC [ADP80])
TSP (G, c)	$> f(\langle G, c \rangle)$, para toda função f computável em tempo polinomial [SG76]

Tabela 8.1: Limiares de aproximação de alguns problemas.

Exercícios

8.1 Verifique que os problemas ESCALONAMENTO, EMPACOTAMENTO, MAXCUT, MAXSAT, MOCHILA, MINCC, MINCV, MINFS, MINMCUT, MINTC, TSPM, e TSP estão em NPO.

8.2 Prove que o problema ESCALONAMENTO (m, n, t) com apenas duas máquinas (ou seja, com $m = 2$) é NP-difícil.

8.3 Uma **estrela** é uma árvore em que no máximo um dos vértices tem grau maior que 1. O problema MINMCUT é NP-difícil mesmo quando restrito a estrelas, ou seja, mesmo para instâncias (G, S, c) em que G é uma estrela. Descreva uma redução do MINCV para o MINMCUT em estrelas. O que se pode dizer sobre algoritmos de aproximação para estes dois problemas?

8.4 Cada problema de otimização pode ser associado de maneira natural a um **problema de decisão** sobre o mesmo conjunto de instâncias. Se Π é um problema de minimização, o problema de decisão associado é

Problema $\Pi_{\mathrm{D}}(I, \gamma)$: *Dados uma instância* I *e um número racional* γ*, decidir se existe* S *em* $\mathrm{Sol}(I)$ *tal que* $\mathrm{val}(I, S) \leq \gamma$. Π_{D}

Em caso de Π ser um problema de maximização, basta trocar $\mathrm{val}(I, S) \leq \gamma$ por $\mathrm{val}(I, S) \geq \gamma$ na definição de Π_{D}. Mostre que se Π está em NPO, então Π_{D} pode ser reduzido a Π.

8.5 Associado a cada problema de otimização Π temos o seguinte **problema de valoração**:

Problema $\Pi_{\mathrm{V}}(I)$: *Dada uma instância* I *de* Π*, calcular* $\mathrm{opt}(I)$. Π_{V}

Mostre que se Π é um problema em NPO, então Π_{V} está em NPO e Π_{V} pode ser reduzido a Π.

8.6 Mostre que se Π é um problema em NPO, então Π_{D} e Π_{V} são polinomialmente equivalentes [ACG+99].

8.7 Seja Π um problema de minimização em NPO tal que $\Pi_{\mathrm{D}}(\cdot, \gamma)$ é NP-completo para algum γ fixo. Mostre que se existe uma $((\gamma + 1)/\gamma)$-aproximação polinomial para Π, então P = NP. Baseado neste fato, o que se pode dizer sobre o problema EMPACOTAMENTO?

8.8 Mostre que os problemas EMPACOTAMENTO e EMPACOTAMENTOINT, definido a seguir, são polinomialmente equivalentes e que uma α-aproximação polinomial para um deles pode ser transformada em uma α-aproximação polinomial para o outro.

Problema EMPACOTAMENTOINT (γ, n, v): *Dados inteiros* *positivos γ e n e, para cada i em $\{1, \ldots, n\}$, um número inteiro v_i em $\{0, \ldots, \gamma\}$, encontrar uma partição $\mathcal{B}$ de $\{1, \ldots, n\}$ tal que $v(B) \leq \gamma$ para todo B em $\mathcal{B}$ e $|\mathcal{B}|$ seja mínimo.*

8.9 Mostre que os problemas ESCALONAMENTO e ESCALONAMENTOINT são polinomialmente equivalentes e que uma α-aproximação polinomial para um dos problemas pode ser transformada em uma α-aproximação polinomial para o outro.

8.10 Verifique que os problemas EMPACOTAMENTOINT, ESCALONAMENTOINT, MAXSAT, MAXCUT e MINCV satisfazem as hipóteses do teorema 8.7.

8.11 Prove o teorema 8.11.

8.12 Prove o teorema 8.12.

8.13 Mostre que se existe uma $f(\langle G, c\rangle)$-aproximação polinomial para o TSP(G, c), então P = NP, onde f é uma função computável em tempo polinomial.

Notas bibliográficas

A apresentação deste capítulo baseou-se nos livros de Ausiello *et al.* [ACP95] e de Papadimitriou [Pap94], bem como no texto de Steger [Ste01] e no livro de Garey e Johnson [GJ75].

Logo após a definição das classes P e NP, já se sabia que a obtenção de aproximações para certos problemas de otimização é tão difícil quanto a obtenção da solução exata [GJ79]:

> *se, para alguma constante α, existe um algoritmo polinomial A tal que* $\mathrm{opt}(I) - \mathrm{val}(I, A(I)) \leq \alpha$ *para toda instância I do problema* MOCHILA, *então* P = NP.

Considerando as classes de complexidade definidas neste capítulo, o teorema acima mostra que o melhor que se pode afirmar do problema MOCHILA é que ele está em FPTAS, já que é improvável que o problema esteja em PO.

Como vimos, usando definições que surgiram posteriormente, o teorema de Sahni e Gonzalez [SG76] pôde ser escrito na demonstração do teorema 8.5 como: *se* TSP *está em* APX, *então* P = NP.

Para outros problemas, resultados similares apareceram de forma isolada, indicando a impossibilidade de certas aproximações. Papadimitriou e Yannakakis [PY91] definiram duas classes de problemas de otimização: as classes MAXNP e MAXSNP. Eles definiram a chamada L-redução entre problemas de otimização e provaram a existência de problemas completos para a classe MAXSNP. A característica dessas classes é que se existir um esquema de aproximação polinomial para um problema completo de uma das classes, esse algoritmo pode ser adaptado para obter um esquema de aproximação polinomial para todos os outros problemas nessa classe. Como corolário dos resultados relativos a provas verificáveis probabilisticamente, Arora, Lund, Motwani, Sudan e Szegedy [ALM+92] mostraram que se os problemas completos para a classe MAXSNP podem ser arbitrariamente aproximados, então P = NP. Hougardy, Prömel e Steger [HPS94], Kohayakawa e Soares [KS95], e Mayr, Prömel e Steger [MPS98] apresentaram um tratamento formal de provas verificáveis probabilisticamente e resultados de inaproximabilidade derivados de tais provas.

MAXNP
MAXSNP

Mais recentemente, Ausiello, Crescenzi e Protasi [ACP95] definiram de forma diferente classes de problemas de otimização e respectivas reduções que se mostraram, de alguma forma, equivalentes às mencionadas acima [KMSV99]. Essas definições são as usadas no presente texto.

É sabido que o problema MinMCut é APX-díficil [DJP+94]; entretanto, não se sabe se ele é APX-completo, ou seja, não se sabe se MinMCut está em APX.

Papadimitriou e Vempala [PV00] anunciaram recentemente que, sob a hipótese de P $\neq$ NP, o limiar de aproximação do TSPM não é inferior a 129/128.

Um compêndio de problemas de otimização, com indicação da classe de complexidade a que pertencem, pode ser encontrado em Crescenzi e Kann [CK00]. Este compêndio faz parte do livro de Ausiello, Crescenzi, Gambosi, Kann, Marchetti-Spaccamela e Protasi [ACG+99].

APÊNDICE A

Teoria dos Grafos

Este apêndice é um resumo da terminologia e notação básicas da teoria dos grafos. A notação é consistente com a de textos clássicos [BM76, Die00] sobre o assunto.

Um **grafo** é um par (V, E), onde V é um conjunto finito arbitrário e E um conjunto de pares não-ordenados[1] de elementos de V. Os elementos de V são chamados **vértices** e os de E são **arestas** (*edges*). Se vw é uma aresta, os vértices v e w são os **extremos** da aresta. Dois vértices v e w de um grafo são **adjacentes** se existe uma aresta com extremos v e w. Se cada vértice é adjacente a todos os outros, então o grafo é **completo**.

O conjunto de vértices de um grafo G é denotado por V_G e o conjunto de arestas por E_G. Se G é completo, então $|E_G| = \frac{1}{2}n(n-1)$, onde $n := |V_G|$. O **tamanho** (veja apêndice E) de um grafo G é o número $\langle G \rangle := |V_G| + |E_G|$.

V_G E_G $\langle G \rangle$

Cortes e graus

Para qualquer conjunto X de vértices de um grafo G, denotamos por $\delta_G(X)$, ou simplesmente por $\delta(X)$, o conjunto de todas as arestas que têm um extremo em X e outro em $V_G \setminus X$. Um **corte** (*cut*) é qualquer conjunto da forma $\delta(X)$, onde X é um subconjunto de V_G. Um conjunto X de vértices **separa** um vértice x de outro y se $x \in X$ enquanto $y \in V_G \setminus X$. Nessas condições, dizemos também que o corte $\delta(X)$ separa x de y.

$\delta(X)$

[1] Um par não-ordenado é um conjunto com exatamente dois elementos. Um par não-ordenado como $\{v, w\}$ é denotado, indiferentemente, por vw ou wv.

Se v é um vértice, escrevemos $\delta(v)$ no lugar de $\delta(\{v\})$. O número $|\delta(v)|$ é o **grau** de v. A soma dos graus de todos os vértices de um grafo é igual ao dobro do número de arestas: $\sum_{v \in V_G} |\delta(v)| = 2|E_G|$.

Um vértice é **isolado** se tem grau 0. O **grau máximo** em G é o
Δ número $\Delta(G) := \max_v |\delta(v)|$.

Subgrafos

Um grafo H é **subgrafo** de um grafo G se $V_H \subseteq V_G$ e $E_H \subseteq E_G$. Se uma das inclusões é própria, H é subgrafo **próprio** de G. Se $V_H = V_G$, dizemos que H é um subgrafo **gerador** (*spanning subgraph*) de G.

$G - F$ Para qualquer conjunto F de arestas de G, denotamos por $G - F$ o subgrafo gerador de G cujo conjunto de arestas é $E_G \setminus F$. Se $F = \{f\}$, podemos escrever $G - f$.

Dizemos que H é um subgrafo **induzido** de G se E_H é o conjunto de todas as arestas de G que têm ambos os extremos em V_H. O subgrafo
$G[X]$ induzido de G que tem X por conjunto de vértices é denotado por $G[X]$.
$G - Y$ Para qualquer subconjunto Y de V_G, denotamos por $G - Y$ o grafo $G[V_G \setminus Y]$. Se $Y = \{y\}$, podemos escrever $G - y$.

O subgrafo de G **induzido** por um subconjunto F de E_G é o grafo $(V(F), F)$, onde $V(F)$ é o conjunto de todos os vértices que são extremo
$G[F]$ de algum elemento de F. Podemos denotar esse subgrafo por $G[F]$.

Se F é um conjunto de pares não-ordenados de elementos de V_G,
$G + F$ denotamos por $G + F$ o grafo $(V_G, E_G \cup F)$ de G. Se $F = \{f\}$, podemos escrever $G + f$.

Passeios, ciclos, caminhos e circuitos

Um **passeio** (*walk*) é uma seqüência $(v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k)$ em que $v_0, \ldots, v_k$ são vértices e cada e_i é uma aresta com extremos v_{i-1} e v_i. Dizemos que v_0 é a **origem** e que v_k é o **término** do passeio. Para simplificar, podemos omitir as arestas e representar um passeio por sua seqüência de vértices, $(v_0, v_1, \ldots, v_k)$. O passeio é **fechado** se $v_k = v_0$ e **aberto** em caso contrário.

Uma **trilha** é um passeio cujas arestas são distintas duas a duas, ou seja, um passeio sem arestas repetidas. Um **caminho** (*path*) é um passeio cujos vértices são distintos dois a dois, ou seja, um passeio sem vértices repetidos. A origem e o término do caminho são os seus **extremos**.

Um **ciclo** (*cycle*) é uma trilha fechada com pelo menos três arestas.

Um **circuito** é um ciclo $(v_0, v_1, \ldots, v_k)$ tal que $v_i \neq v_j$ sempre que $i < j$, exceto se $i = 0$ e $j = k$.

Um passeio é **gerador** se contém todos os vértices do grafo. É claro que a definição se aplica, em particular, a trilhas, caminhos, ciclos e circuitos. Um caminho gerador é também conhecido como **caminho hamiltoniano**. Analogamente, um circuito gerador é conhecido como **circuito hamiltoniano**.

O conjunto de arestas de um caminho ou circuito C é denotado por E_C. O **comprimento** de um caminho ou circuito C é $|E_C|$, ou seja, o número de arestas de C. E_C

Conexão e componentes

Um grafo é **conexo** se, para todo par (v, w) de seus vértices, existe um passeio com origem v e término w. É claro que um grafo G é conexo se e somente se $\delta_G(X) \neq \emptyset$ para todo subconjunto não vazio e próprio X de V_G.

Um **componente** de um grafo G é qualquer subgrafo conexo maximal[2] de G. Às vezes convém confundir um componente com o seu conjunto de vértices.

Florestas e árvores

Uma **floresta** é um grafo sem ciclos. Se F é uma floresta, então $|E_F| = |V_F| - c(F)$, onde $c(F)$ é o número de componentes de F.

Uma **árvore** (*tree*) é uma floresta conexa. Portanto, cada componente de uma floresta é uma árvore. Se T é uma árvore, então $|E_T| = |V_T| - 1$. Para qualquer par (x, y) de vértices em uma árvore, existe um e um só caminho de x a y.

Uma **floresta geradora** de um grafo G é qualquer subgrafo gerador de G que seja uma floresta, isto é, qualquer floresta F tal que $V_F = V_G$ e $E_F \subseteq E_G$. Uma **árvore geradora** (*spanning tree*) de um grafo G é qualquer subgrafo gerador de G que seja uma árvore. Todo grafo conexo tem uma árvore geradora.

---

[2] Um subgrafo H de G é **maximal** com relação a uma dada propriedade se H não é subgrafo próprio de um outro subgrafo H' de G que tenha a propriedade em questão. Subgrafos **minimais** são definidos de maneira análoga.

Grafos bipartidos

Uma **bipartição** de um grafo G é qualquer partição $\{X, Y\}$ de V_G tal que $\delta(X) = E_G$. Um grafo admite uma bipartição se e somente se não tem circuito de comprimento ímpar [BM76]. Um **grafo bipartido** (*bipartite graph*) é um grafo munido de uma bipartição.

Grafos planares

Um grafo é **planar** se for isomorfo a um mapa. Mais precisamente, um grafo G é planar se existe um mapa (V, E) e uma bijeção de ψ de V_G em V tal que, para todo par (v, w) de vértices de G, tem-se $vw \in E_G$ se e somente se algum elemento de E tem extremos $\psi(v)$ e $\psi(w)$.

Um **mapa** é um par (V, E) onde (1) V é um conjunto finito de pontos do plano, (2) todo elemento de E é um arco entre dois elementos de V, (3) elementos diferentes de E têm diferentes conjuntos de extremos, (4) o interior de um elemento de E não contém elementos de V nem pontos que pertençam a outro elemento de E.

Um **arco** é a união de um número finito de segmentos de reta no plano que seja homeomorfa ao intervalo fechado $[0, 1]$ da reta; as imagens de 0 e 1 sob esse homeomorfismo são os **extremos** do arco.

Multigrafos

Às vezes, convém admitir a existência de arestas múltiplas em um grafo, ou seja, a existência de várias "cópias" de uma mesma aresta. Dizemos nesse caso que temos um multigrafo. Mais explicitamente, um **multigrafo** consiste em um conjunto finito V e um multiconjunto[3] E de pares não-ordenados de elementos de V. A terminologia e notação definidas para grafos se estendem aos multigrafos com as devidas adaptações. Assim, por exemplo, o grau de um vértice é a somas das multiplicidades das arestas que nele incidem. Outro exemplo: ciclos e circuitos podem ter apenas duas arestas (mas não menos que isso).

Custos e pesos

Suponha que c é uma função que associa um número c_e a cada aresta e de um grafo. Para qualquer conjunto F de arestas do grafo, denotamos

[3] Um multiconjunto pode ter mais de uma cópia de cada um de seus elementos; o número de cópias é a *multiplicidade* do elemento.

por $c(F)$ a soma dos c_f para f em F: $c(\,)$

$$c(F) := \textstyle\sum_{f \in F} c_f \,.$$

Se $F = \emptyset$, então $c(F) = 0$. Se H é um subgrafo do grafo em questão, definimos $c(H)$ como $c(E_H)$. Se c_e é interpretado como **custo** da aresta e, dizemos que $c(F)$ é o custo de F e $c(H)$ o custo de H. Algo análogo acontece quando c_e é interpretado como **peso** da aresta e.

Apêndice B

Vetores e Matrizes

Um **vetor** é uma função que leva um conjunto finito arbitrário — o conjunto de índices — em um conjunto de números reais. Se o conjunto de índices de um vetor x é N, dizemos que x está definido sobre N ou é indexado por N. Se x é um vetor sobre um conjunto N e j é um elemento de N então x_j denota o componente j de x, isto é, o valor da função x em j. Se Q é um subconjunto de N então x_Q denota a restrição de x a Q, ou seja, o vetor cujo componente q é x_q para cada q em Q. x_Q

Se todos os componentes são números racionais, dizemos que o vetor é **racional**. Se forem todos inteiros, dizemos que o vetor é **inteiro**. Como nossos interesses são computacionais, quase todos os vetores do texto são racionais (a exceção fica no capítulo 7, que trata de programação semidefinida).

Um vetor x é **nulo** se $x_j = 0$ para cada índice j; o vetor nulo será denotado por 0 qualquer que seja o seu conjunto de índices. Se x e y são vetores sobre um mesmo conjunto de índices e $x_j \geq y_j$ para cada j, dizemos que $x \geq y$. A relação $\leq$ é definida de modo análogo. $x \geq y$

O **vetor característico** de um subconjunto P de N é um vetor x sobre N tal que $x_i = 1$ se $i \in P$ e $x_i = 0$ em caso contrário. $x \geq 0$

Matrizes

Uma **matriz** é uma função que leva o produto cartesiano de dois conjuntos finitos em um conjunto de números reais. Se uma matriz A tem domínio $M \times N$, dizemos que M é o conjunto de índices de linhas e N o conjunto de índices de colunas de A. Dizemos também que A é uma matriz sobre $M \times N$ ou indexada por $M \times N$. Se i e j são elementos de M e N, respectivamente, então A_{ij} denota o componente (i, j) de A, ou seja, o valor de A em (i, j).

Uma **linha** de A é um *vetor* sobre N: a linha i de A é o vetor cujo componente j é A_{ij} para cada j em N. A **coluna** j de A é definida
$A_{\star j}$ analogamente. A coluna j de A será denotada por A_{Mj} ou $A_{\star j}$ e a linha
$A_{i\star}$ i por A_{iN} ou $A_{i\star}$.

A_{PQ} Se P e Q são subconjuntos de M e N, respectivamente, então A_{PQ} é a restrição de A a $P \times Q$, ou seja, a matriz sobre $P \times Q$ cujo componente (p, q) vale A_{pq} para cada p em P e cada q em Q.

Dizemos que a matriz é **racional** se todos os seus componentes são racionais e **inteira** se todos os seus componentes são inteiros.

$A^\top$ A **transposta** de uma matriz A sobre $M \times N$ é a matriz $A^\top$ sobre $N \times M$ definida por $A^\top_{ij} = A_{ji}$. Portanto, a linha i de $A^\top$ é a coluna i de A. É claro que a transposta de $A^\top$ é A. Uma matriz A é **simétrica** se $A^\top = A$.

Uma matriz é **quadrada** se seus conjuntos de índices de linhas e colunas são iguais. É claro que toda matriz simétrica é quadrada. Uma matriz D é **diagonal** se for quadrada e se $D_{ij} = 0$ sempre que $i \neq j$. É evidente que toda matriz diagonal é simétrica.

Uma matriz **identidade** é uma matriz diagonal I tal que $I_{ii} = 1$
I para cada i. Matrizes identidade são denotadas por I, quaisquer que sejam seus conjuntos de índices.

Produtos

Para quaisquer vetores x e y sobre um mesmo conjunto N, o produto (interno) de x por y é o número

$$xy \ := \ \textstyle\sum_{j \in N} x_j y_j \, .$$

É óbvio que $xy = yx$. Por exemplo, se y é o vetor característico de um subconjunto Q de N, então xy é a soma $\sum_{q \in Q} x_q$, que convencionamos denotar por $x(Q)$.

Suponha agora que A é uma matriz sobre $M \times N$. O produto de A por um vetor x sobre N é o vetor

$$Ax$$

sobre M cujo componente i é o produto da linha i de A por x, ou seja, o número $\sum_j A_{ij} x_j$. O produto de um vetor y sobre M por A é o vetor

$$yA$$

cujo componente j é o número $\sum_i y_i A_{ij}$. Em suma, $(Ax)_i = \sum_j A_{ij} x_j$ e $(yA)_j = \sum_i y_i A_{ij}$, ou seja, yA é uma combinação linear das linhas de A enquanto Ax é uma combinação linear das colunas de A.

É evidente que $yA = A^\top y$. É menos evidente, mas não menos verdadeira, a propriedade associativa $y(Ax) = (yA)x$, ou seja, a inversão da ordem das somas:

$$\sum_i y_i (\sum_j A_{ij} x_j) = \sum_j (\sum_i y_i A_{ij}) x_j \,.$$

Para qualquer matriz A sobre $L \times M$ e qualquer matriz B sobre $M \times N$, o produto de A por B é a matriz

$$AB$$

sobre $L \times N$ cujo componente (i, k) é o produto de vetores $A_{i\star} B_{\star k}$. É evidente que $(AB)^\top = B^\top A^\top$. É menos evidente que $(AB)C = A(BC)$, desde que cada produto faça sentido.

Uma matriz quadrada A é **inversível** se existe uma matriz B tal que $AB = BA = I$. Uma tal matriz B é **inversa** de A.

Normas e tamanhos

Denotamos por $\|x\|$ a **norma** do vetor x. Assim, $\|x\| := \sqrt{xx}$. $\|x\|$
Esse número pode ser interpretado como o comprimento geométrico do vetor x.

O **tamanho** de um número inteiro é o número de caracteres (digamos dígitos decimais) na representação do número (veja apêndice E). O tamanho de um número inteiro α é denotado por $\langle\alpha\rangle$. O tamanho de $\langle\alpha\rangle$
um número racional com numerador α e denominador β é a soma dos tamanhos de α e β. O tamanho de α/β é denotado por $\langle\alpha/\beta\rangle$. $\langle\alpha/\beta\rangle$

O **tamanho** de um vetor ou matriz racionais é a soma dos tamanhos de seus componentes. É claro que o tamanho de uma matriz não é inferior ao número de componentes da matriz. O tamanho de uma matriz A é denotado por $\langle A\rangle$. $\langle A\rangle$

O **tamanho** de um conjunto de vetores e matrizes racionais é a soma dos tamanhos dos elementos do conjunto. Assim, o tamanho de um sistema (A, b, c) é $\langle A\rangle + \langle b\rangle + \langle c\rangle$.

Essas definições de tamanho são casos particulares do conceito geral de tamanho a que se refere a seção 1.2 e o apêndice E.

Matrizes positivas semidefinidas

A caracterização de matrizes positivas semidefinidas que descrevemos a seguir é usada no capítulo 7.

Uma matriz quadrada X é **positiva semidefinida** se existe uma matriz quadrada Y tal que

$$X = YY^\top .$$

É claro que toda matriz positiva semidefinida é simétrica. O seguinte lema caracteriza tais matrizes.

> **Lema B.1**: *Para toda matriz simétrica X vale uma e apenas uma das seguintes alternativas: existe uma matriz quadrada Y tal que $X = YY^\top$ ou existe um vetor y tal que $yXy < 0$.*

A prova do lema depende de alguns conceitos que passamos a definir. Uma matriz quadrada é **elementar** se coincide com a identidade na diagonal e em todas as colunas, exceto talvez uma. Mais precisamente, uma matriz E indexada por $M \times M$ é elementar se existe m em M tal que $E_{mm} = 1$ e $E_{\star j} = I_{\star j}$ para cada j distinto de m. Toda matriz elementar E é inversível e sua inversa, digamos H, também é elementar: $H_{mm} = 1$ e $H_{im} = -E_{im}$ para cada i distinto de m e $H_{\star j} = I_{\star j}$ para cada j distinto de m.

Um **produto de matrizes elementares** é qualquer matriz da forma $E^{(1)} \cdots E^{(p)}$, onde cada $E^{(i)}$ é uma matriz elementar indexada por $M \times M$. Se $H^{(i)}$ é a inversa de $E^{(i)}$ para cada i então o produto de matrizes elementares $H^{(p)} \cdots H^{(1)}$ é a inversa de $E^{(1)} \cdots E^{(p)}$.

Demonstração de B.1: A prova de que apenas uma das alternativas vale é simples: se $X = YY^\top$ então

$$yXy = y(YY^\top)y = (yY)(Y^\top y) = (yY)(yY) \geq 0 .$$

Para provar que uma das alternativas vale, vamos descrever um algoritmo que recebe qualquer matriz simétrica X e devolve uma matriz real Y tal que $X = YY^\top$ ou um vetor y tal que $yXy < 0$.

Para qualquer matriz simétrica X indexada por $M \times M$ e qualquer m em M tal que $X_{mm} \neq 0$, é fácil determinar uma matriz elementar E tal que a matriz simétrica $Z := EXE^\top$ tem a seguinte estrutura:

$$Z_{mm} = X_{mm} \quad \text{e} \quad Z_{mj} = Z_{jm} = 0$$

$$
\begin{array}{c|ccc|ccc|}
 & & P & & & M \setminus P & \\
\hline
 & -2 & 0 & 0 & 0 & 0 & 0 \\
P & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 3 & 0 & 0 & 0 \\
\hline
 & 0 & 0 & 0 & 0 & & \\
M \setminus P & 0 & 0 & 0 & & 0 & \\
 & 0 & 0 & 0 & & & 0 \\
\hline
\end{array}
$$

Figura B.1: Estrutura da matriz Z.

para cada j distinto de m. Basta tomar como E a matriz elementar definida por $E_{mm} = 1$, $E_{im} = -X_{im}/X_{mm}$ para cada i distinto de m e $E_{\star j} = I_{\star j}$ para cada j distinto de m.

Segue daí que para qualquer matriz simétrica X indexada por $M \times M$ existe um produto F de matrizes elementares tal que a matriz simétrica

$$Z \;:=\; FXF^{\top}$$

tem a seguinte estrutura: existe um subconjunto P de M tal que Z_{PP} é uma matriz diagonal, $Z_{P,M \setminus P} = 0$ e, para cada m em $M \setminus P$, $Z_{mm} = 0$ mas $Z_{m\star} \neq 0$. (Veja figura B.1.)

Se $P = M$ e $Z_{pp} \geq 0$ para cada p em P então Z é diagonal e $X = GZG^{\top}$, onde G é a inversa de F. Portanto,

$$X \;=\; YY^{\top}\,,$$

onde $Y = G\dot{Z}$ e $\dot{Z}$ é a matriz diagonal definida por $\dot{Z}_{pp} = \sqrt{Z_{pp}}$. Suponha agora que $Z_{mm} < 0$ para algum m em M. Seja u o vetor definido por $u_m = 1$ e $u_i = 0$ para cada i distinto de m. Então $(uF)X(uF) = u(FXF^{\top})u = uZu = Z_{mm} < 0$. Em suma,

$$yXy \;<\; 0\,,$$

para $y = uF$. Finalmente, suponha que $P \neq M$ e seja m um elemento de $M \setminus P$. Então $Z_{mm} = 0$ mas $Z_{mj} \neq 0$ para algum j. Seja u o vetor definido da seguinte maneira: $u_j = 1$, $u_i = 0$ para i distinto de m e j e u_m é escolhido de modo que $2u_m Z_{mj} + Z_{jj} < 0$. Então $(uF)X(uF) = uZu < 0$ e portanto

$$yXy \;<\; 0\,,$$

para $y = uF$. $\Box$

Convém observar que a matriz Y no lema B.1 poderá não ser racional mesmo que a matriz X seja racional.

Algoritmo POSSEMIDEF (X)

```
1   $Z \leftarrow X$
2   $F \leftarrow G \leftarrow I$
3   para $m$ de 1 a $n$ faça
4       se $Z_{mm} \neq 0$
5           então $E \leftarrow H \leftarrow I$
6                 para $i$ de 1 a $n$ faça $E_{im} \leftarrow -Z_{im}/Z_{mm}$
7                 para $i$ de 1 a $n$ faça $H_{im} \leftarrow -E_{im}$
8                 $E_{mm} \leftarrow H_{mm} \leftarrow 1$
9                 $Z \leftarrow EZE^\top$
10                $F \leftarrow EF$
11                $G \leftarrow GH$    $\triangleright$  $FG = GF = I$
12  para $m$ de 1 a $n$ faça
13      se $Z_{mm} < 0$
14          então $u \leftarrow 0$
15                $u_m \leftarrow 1$
16                devolva $uF$
17      se $Z_{mm} = 0$
18          então $j \leftarrow 1$
19                enquanto $j \leq n$ e $Z_{mj} = 0$ faça $j \leftarrow j + 1$
20                se $j \leq n$
21                   então $u \leftarrow 0$
22                         $u_j \leftarrow 1$
23                         $u_m \leftarrow -Z_{jj}/2Z_{mj}$
24                         se $Z_{mj} > 0$
25                            então $u_m \leftarrow u_m - 1$
26                            senão $u_m \leftarrow u_m + 1$
27                         devolva $uF$
28  para $i$ de 1 a $n$ faça $Z_{ii} \leftarrow \sqrt{Z_{ii}}$
29  devolva $GZ$
```

Figura B.2: O algoritmo POSSEMIDEF recebe uma matriz quadrada X indexada por $\{1, \ldots, n\} \times \{1, \ldots, n\}$ e devolve uma matriz quadrada Y tal que $X = YY^\top$ ou um vetor y tal que $yXy < 0$. Se a matriz X é racional, o vetor y é racional e a matriz Y é racional exceto pela raiz quadrada na linha 28. Nessas condições, o consumo de tempo do algoritmo é polinomial, exceto pela raiz quadrada.

Programação Linear

Este apêndice é um resumo da terminologia e de alguns fatos básicos de programação linear [Pad99, Chv83, Sch86, Feo01]. Veja nossas convenções de notação no apêndice B.

Problema de programação linear

Um **problema de programação linear**, ou **programa linear**, ou simplesmente **pl**, consiste no seguinte: Dada uma matriz A indexada por $M \times N$, um vetor b indexado por M, um vetor c indexado por N e partições[1] $\{M_1, M_2, M_3\}$ e $\{N_1, N_2, N_3\}$ de M e N respectivamente, encontrar um vetor x indexado por N que

$$
\begin{array}{rrcll}
\text{minimize} & cx & & & \\
\text{sob as restrições} & (Ax)_i & \geq & b_i & \text{para cada } i \text{ em } M_1\,, \\
& (Ax)_i & = & b_i & \text{para cada } i \text{ em } M_2\,, \\
& (Ax)_i & \leq & b_i & \text{para cada } i \text{ em } M_3\,, \\
& x_j & \geq & 0 & \text{para cada } j \text{ em } N_1\,, \\
& x_j & \leq & 0 & \text{para cada } j \text{ em } N_3\,.
\end{array}
\tag{C.1}
$$

Usaremos a abreviatura $\mathrm{P}(A, b, c)$ para denotar esse pl; a seqüência de conjuntos $M_1, M_2, M_3, N_1, N_2, N_3$ fica subentendida nessa notação. Embora o pl tenha sido formulado como um problema de minimização, nossa definição inclui, implicitamente, problemas de maximização, uma vez que minimizar cx é o mesmo que maximizar $-cx$. $\mathrm{P}(A, b, c)$

[1] Ao contrário da definição usual, qualquer das partes de nossas partições pode ser vazia.

A tradição impõe uma terminologia que, infelizmente, desrespeita o significado da palavra "solução". Assim, uma **solução viável** (*feasible solution*) do problema é qualquer vetor x que satisfaz as restrições; e uma **solução ótima** é qualquer solução viável x que minimize cx.

$X(A,b)$

O conjunto de todas as soluções viáveis do problema $\mathrm{P}(A,b,c)$ será denotado por $X(A,b)$. Se $X(A,b)$ é vazio, dizemos que o problema é **inviável** (*infeasible*). Caso contrário, o problema é **viável** (*feasible*). Dizemos que o problema é **limitado** (*bounded*) se existe um número ω tal que $cx \geq \omega$ para todo x em $X(A,b)$; caso contrário, o problema é **ilimitado** (*unbounded*). É claro que problemas inviáveis e ilimitados não têm solução ótima.

Dualidade

Há uma importante relação de dualidade entre pls. O **dual** do pl $\mathrm{P}(A,b,c)$ é o pl $\mathrm{P}(-A^\top, -c, -b)$, onde $A^\top$ é a transposta de A e os conjuntos de índices são $N_1, N_2, N_3, M_1, M_2, M_3$. É claro que esse pl também pode ser escrito assim: encontrar um vetor y indexado por M que

$$
\begin{array}{rrcll}
\text{maximize} & y\,b & & & \\
\text{sob as restrições} & (yA)_j & \leq & c_j & \text{para cada } j \text{ em } N_1\,, \\
& (yA)_j & = & c_j & \text{para cada } j \text{ em } N_2\,, \\
& (yA)_j & \geq & c_j & \text{para cada } j \text{ em } N_3\,, \\
& y_i & \geq & 0 & \text{para cada } i \text{ em } M_1\,, \\
& y_i & \leq & 0 & \text{para cada } i \text{ em } M_3\,.
\end{array}
\tag{C.2}
$$

$\mathrm{D}(A,c,b)$ $Y(A,c)$

Convém usar uma notação específica para o dual: o pl (C.2) será denotado por $\mathrm{D}(A,c,b)$ e seu conjunto de soluções viáveis por $Y(A,c)$.

Em discussões sobre um par dual de pls, é comum dizer que um deles — digamos (C.1) — é "o primal" e o outro — digamos (C.2) — é "o dual". Adotada essa convenção, cada componente de um vetor x indexado por N é uma "variável primal" e cada componente de um vetor y indexado por M é uma "variável dual".

Problemas canônicos

Se $M_1 = M$ e $N_1 = N$, o problema $\mathrm{P}(A,b,c)$ se reduz a encontrar um vetor x que

$$
\begin{array}{rrcl}
\text{minimize} & cx & & \\
\text{sob as restrições} & Ax & \geq & b\,, \\
& x & \geq & 0\,.
\end{array}
\tag{C.3}
$$

Diz-se, às vezes, que esse é o pl **canônico primal**, pois qualquer pl é equivalente a um pl que tem essa forma, e portanto qualquer algoritmo que resolva o pl canônico primal poderia ser usado para resolver um pl arbitrário. É claro que poderíamos enunciar (C.3), sem notação matricial, dizendo

$$\begin{array}{rrcll} \text{minimize} & \sum_j c_j x_j & & & \\ \text{sob as restrições} & \sum_j A_{ij} x_j & \geq & b_i & \text{para cada } i\,, \\ & x_j & \geq & 0 & \text{para cada } j\,. \end{array}$$

O dual do pl (C.3) consiste em encontrar um vetor y que

$$\begin{array}{rrcl} \text{maximize} & y\,b & & \\ \text{sob as restrições} & y\,A & \leq & c\,, \\ & y & \geq & 0\,. \end{array} \tag{C.4}$$

Pode-se dizer que esse é o pl **canônico dual**.

Lema da dualidade

Há uma relação fundamental entre as soluções viáveis de um pl e as soluções viáveis de seu dual.

Lema C.1 (da dualidade): *Para todo x em $X(A,b)$ e todo y em $Y(A,c)$ tem-se $cx \geq y\,b$.*

Demonstração: Se $M_1 = M$ e $N_1 = N$ então as restrições de (C.3) e (C.4) garantem as desigualdades $cx \geq (y\,A)x = y\,(Ax) \geq y\,b$. A primeira vale porque $c \geq y\,A$ e $x \geq 0$; a última, porque $y \geq 0$ e $Ax \geq b$. A demonstração do caso geral é conceitualmente análoga mas tipograficamente indigesta: em virtude das restrições de (C.1) e (C.2),

$$\begin{array}{rcl} cx & = & c_{N_1} x_{N_1} + c_{N_2} x_{N_2} + c_{N_3} x_{N_3} \\ & \geq & (y\,A)_{N_1} x_{N_1} + (y\,A)_{N_2} x_{N_2} + (y\,A)_{N_3} x_{N_3} \\ & = & (y_{M_1} A_{M_1 N_1} + y_{M_2} A_{M_2 N_1} + y_{M_3} A_{M_3 N_1}) x_{N_1} + \\ & & (y_{M_1} A_{M_1 N_2} + y_{M_2} A_{M_2 N_2} + y_{M_3} A_{M_3 N_2}) x_{N_2} + \\ & & (y_{M_1} A_{M_1 N_3} + y_{M_2} A_{M_2 N_3} + y_{M_3} A_{M_3 N_3}) x_{N_3} \\ & = & y_{M_1} (A_{M_1 N_1} x_{N_1} + A_{M_1 N_2} x_{N_2} + A_{M_1 N_3} x_{N_3}) + \\ & & y_{M_2} (A_{M_2 N_1} x_{N_1} + A_{M_2 N_2} x_{N_2} + A_{M_2 N_3} x_{N_3}) + \\ & & y_{M_3} (A_{M_3 N_1} x_{N_1} + A_{M_3 N_2} x_{N_2} + A_{M_3 N_3} x_{N_3}) \\ & = & y_{M_1} (Ax)_{M_1} + y_{M_2} (Ax)_{M_2} + y_{M_3} (Ax)_{M_3} \end{array}$$

$$
\begin{aligned}
&\geq \; y_{M_1} b_{M_1} + y_{M_2} b_{M_2} + y_{M_3} b_{M_3} \\
&= \; y\,b\,,
\end{aligned}
$$

onde $A_{M_1 N_1}$, por exemplo, denota a restrição de A a $M_1 \times N_1$. □

O lema é às vezes chamado, um tanto pomposamente, de *teorema fraco* da dualidade. Ele leva à seguinte observação, óbvia mas importante: para tornar evidente que um certo vetor x em $X(A, b)$ é solução ótima do problema $\mathrm{P}(A, b, c)$, basta exibir um vetor y em $Y(A, c)$ tal que

$$cx = y\,b\,.$$

Com isso, estaremos mostrando também que y é solução ótima do problema $\mathrm{D}(A, c, b)$. Eis outra conseqüência do lema C.1: para mostrar que o problema $\mathrm{P}(A, b, c)$ é limitado, basta exibir um elemento de $Y(A, c)$.

Folgas complementares

A demonstração do lema C.1 sugere o seguinte conceito. Dois vetores x e y, indexados por M e N respectivamente, têm **folgas complementares** (*complementary slackness*) se, para cada j em $N_1 \cup N_3$, temos

$$x_j = 0 \quad \text{ou} \quad (y\,A)_j = c_j \tag{C.5}$$

e, para cada i em $M_1 \cup M_3$, temos

$$y_i = 0 \quad \text{ou} \quad (A\,x)_i = b_i\,. \tag{C.6}$$

Há quem diga que (C.5) dá as "condições de folgas complementares primais" e (C.6) as de "folgas complementares duais".

Lema C.2 (das folgas complementares): *Para todo x em $X(A, b)$ e todo y em $Y(A, c)$, tem-se $cx = y\,b$ se e somente se as folgas de x e y são complementares.*

Demonstração: A demonstração do lema C.1 deixa claro que $cx = y\,b$ se e somente se $cx = (y\,A)x$ e $y\,(A\,x) = y\,b$. Mas $cx = (y\,A)x$ equivale a (C.5) e $y\,(A\,x) = y\,b$ equivale a (C.6). □

Teorema da dualidade

O teorema da dualidade dá as condições em que um pl tem solução ótima e garante que as soluções ótimas do pl e de seu dual têm o mesmo valor:

Teorema C.3 (da dualidade): *Para qualquer sistema (A, b, c), se o problema $\mathrm{P}(A, b, c)$ é viável e limitado então tem solução ótima; mais especificamente, existem x em $X(A, b)$ e y em $Y(A, c)$ tais que $cx = yb$. Se A, b e c forem racionais então podemos supor que x e y também são racionais.*

Esse teorema, também conhecido como *teorema forte* da dualidade, pode ser resumido da seguinte maneira: a menos que os dois pls sejam inviáveis,

$$\min_x cx \;=\; \max_y yb$$

para x variando em $X(A, b)$ e y em $Y(A, c)$. Esta identidade vale mesmo quando um dos pls é inviável, desde que estejamos dispostos a dizer que $\min cx = +\infty$ quando o pl primal é inviável, que $\max yb = +\infty$ quando o pl dual é ilimitado, etc.

Algoritmos de programação linear

O célebre **algoritmo Simplex** resolve qualquer par dual de programas lineares. Ao receber um sistema racional (A, b, c) e os conjuntos $M_1, M_2, M_3, N_1, N_2, N_3$ de índices, o Simplex decide se os problemas $\mathrm{P}(A, b, c)$ e $\mathrm{D}(A, c, b)$ são viáveis e em caso afirmativo produz soluções ótimas racionais, x e y, dos dois programas lineares. Cada componente desses vetores tem numerador e denominador limitados superiormente pelo tamanho do sistema (A, b, c).

Embora o Simplex seja eficiente em média, seu consumo de tempo não é limitado por uma função polinomial do número de componentes do sistema (A, b, c). O consumo de tempo não é limitado nem mesmo por uma função polinomial do tamanho de (A, b, c).

Mas existem algoritmos, muito diferentes do Simplex, capazes de resolver qualquer pl em tempo polinomial.[2] O primeiro algoritmo desse tipo foi publicado por Khachiyan [Kha79, PS82, Sch86, GLS93] e ficou conhecido como **algoritmo dos elipsóides** (*ellipsoid method*). (Outra família de algoritmos polinomiais, conhecidos como **métodos de pontos interiores**, apareceu mais tarde [Kar84, Gon89, Gon92].) Para resolver o problema $\mathrm{P}(A, b, c)$, esse algoritmo consome uma quantidade de tempo limitada por um polinômio no tamanho de (A, b, c).

[2] Esses algoritmos são polinomiais mas não fortemente polinomiais. Veja apêndice E.

> **Fato C.4**: *Existe um algoritmo polinomial que, ao receber um sistema racional (A, b, c), decide se o problema $\mathrm{P}(A, b, c)$ tem solução e, em caso afirmativo, devolve uma solução ótima racional x. O tamanho de x é limitado por $2\langle A\rangle + 2\langle b\rangle$. O consumo de tempo do algoritmo é limitados por um polinômio em $\langle A\rangle + \langle b\rangle + \langle c\rangle$.*

Programação linear e algoritmos de separação

Muitos problemas combinatórios podem ser representados, de maneira aproximada, por problemas da forma $\mathrm{P}(A, b, c)$, com A, b e c racionais. Numa tal classe de programas lineares, o número de linhas de A pode
τ ser uma função exponencial do número de colunas. Digamos que τ é o tamanho da maior linha de (A, b), ou seja, $\tau := \max_i\{\langle A_{i\star}\rangle + \langle b_i\rangle\}$. Mesmo que o número de linhas de A não seja limitado por um polinômio em τ, é possível resolver o problema $\mathrm{P}(A, b, c)$ em tempo limitado por um polinômio em τ e $\langle c\rangle$ se dispusermos de um bom algoritmo para o seguinte

> **Problema da separação**: *Decidir se um dado vetor x está em $X(A, b)$ e, em caso negativo, determinar uma restrição violada, ou seja, uma linha i tal que $(Ax)_i < b_i$.*

Suponha que temos um bom algoritmo para esse problema, isto é, um algoritmo cujo consumo de tempo é limitado por um polinômio em τ e $\langle x\rangle$.[3] Grötschel, Lovász e Schrijver mostraram [GLS93, Sch86] que um tal **algoritmo de separação** pode ser combinado com o algoritmo dos elipsóides de modo a resolver o problema $\mathrm{P}(A, b, c)$ em tempo limitado por uma função polinomial de τ e $\langle c\rangle$ apenas.

Suponha agora que o problema de otimização combinatória que deu origem a $\mathrm{P}(A, b, c)$ tem tamanho σ. Se τ e $\langle c\rangle$ são limitados por um polinômio em σ e temos um bom algoritmo de separação (veja o exercício C.3) então o problema $\mathrm{P}(A, b, c)$ pode ser resolvido em tempo limitado por um polinômio em σ.

---

[3] O consumo de tempo não depende, portanto, do número de linhas do sistema. Veja o exercício C.3.

Problemas de viabilidade

O **problema da viabilidade** consiste em encontrar um vetor viável do problema $\mathrm{P}(A, b, c)$, ou seja, encontrar um elemento de $X(A, b)$. É claro que o problema pode não ter solução. Para certificar a inexistência de solução, basta exibir um **vetor de inviabilidade**, isto é, um vetor y' em $Y(A, 0)$ tal que $y' b > 0$ (veja exercício C.4).

> **Lema C.5** (de Farkas): *Para qualquer sistema (A, b, c), o conjunto $X(A, b)$ é vazio se e somente se existe um vetor y' em $Y(A, 0)$ tal que $y' b > 0$.*

O problema da viabilidade equivale ao caso em que $c = 0$ no problema $\mathrm{P}(A, b, c)$; e esse caso é, às vezes, computacionalmente mais simples que o caso geral. O método primal-dual, descrito no capítulo 5, tira proveito dessa observação para propor um procedimento de resolução de programas lineares.

Exercícios

C.1 Transforme um programa linear arbitrário num programa canônico equivalente.

C.2 Muitos problemas combinatórios envolvem vetores não-negativos b e c e uma matriz A sem componentes negativos e sem linhas nulas. Verifique que, nesse caso, os problemas (C.3) e (C.4) são viáveis. O teorema da dualidade garante então que ambos os problemas têm solução ótima.

C.3 Sejam s e t dois vértices de um grafo G e seja $\mathcal{S}$ a coleção de todos os subconjuntos de V_G que contêm s mas não contêm t. (O número de tais subconjuntos é, tipicamente, uma função exponencial do número $|V_G|$). Seja A a matriz de incidência de $\mathcal{S}$, isto é, a matriz indexada por $\mathcal{S} \times E_G$ cujos componentes são definidos da seguinte maneira: para cada S em $\mathcal{S}$ e cada e em E_G, o componente (S, e) de A vale 1 se $e \in \delta(S)$ e vale 0 em caso contrário. Agora considere o problema de decidir se um dado vetor não-negativo x indexado por E_G satisfaz as restrições

$$Ax \geq \mathit{1}\,, \tag{C.7}$$

onde $\mathit{1}$ denota o vetor cujos componentes são todos iguais a 1. Esboce um algoritmo para o problema. O algoritmo deve receber o grafo G, vértices s e t e um vetor racional não-negativo x e deve devolver 1 se (C.7) estiver satisfeita e 0 em caso contrário. O consumo de tempo de seu algoritmo deve ser limitado por um polinômio em $\langle G\rangle + \langle x\rangle$. *Sugestão:* Limite-se, numa primeira versão, ao caso em que os componentes de x estão todos em $\{0,1\}$.

C.4 Seja y' um elemento de $Y(A,0)$ tal que $y'b > 0$. Mostre que $X(A,b)$ é vazio. Mostre também que se $Y(A,c)$ não é vazio então o problema $\mathrm{D}(A,c,b)$ é ilimitado. Prove afirmações análogas depois de trocar os papéis de $X(A,b)$ e $Y(A,c)$.

Apêndice D

Teoria das Probabilidades

Neste apêndice resumimos os conceitos usados nas análises probabilísticas apresentadas nos capítulos 6 e 7.

Espaços discretos de probabilidade

Uma **medida discreta de probabilidade** sobre um conjunto Ω é uma função $\mathcal{P} : \Omega \to [0,1]$ para a qual $\mathcal{P}(\Omega) := \sum_{\omega\in\Omega} \mathcal{P}(\omega) = 1$. Um **espaço discreto de probabilidade** é um par $(\Omega, \mathcal{P})$ onde Ω é um conjunto finito e não-vazio e $\mathcal{P}$ é uma medida discreta de probabilidade sobre Ω.

Variáveis aleatórias

Uma **variável aleatória** sobre Ω é uma função $X : \Omega \to \{\lambda_1, \dots, \lambda_n\}$, onde cada λ_i é um número real. Um **evento** é o conjunto $X^{-1}(S)$ para algum subconjunto S de $\{\lambda_1, \dots, \lambda_n\}$. Tradicionalmente, um evento $X^{-1}(S)$ é denotado por $[X \in S]$ ou simplesmente por $[X = x]$, se $S = \{x\}$. De forma análoga, podemos usar a notação $[X \neq x]$, $[X < x]$, $[X \leq x]$, $[X > x]$ e $[X \geq x]$.

A probabilidade do evento $[X \in S]$ é o número $\mathcal{P}(X^{-1}(S))$, denotado por $\mathbf{Pr}[X{\in}S]$. A **esperança** (ou valor esperado) da variável aleatória X é o número $\mathbf{Pr}[\,]$ $\mathbf{E}[\,]$

$$\mathbf{E}[X] := \textstyle\sum_{i=1}^{n} \lambda_i \, \mathbf{Pr}[X{=}\lambda_i] \,.$$

Lema D.1 (desigualdade de Markov): *Se $(\Omega, \mathcal{P})$ é um espaço discreto de probabilidade e X é uma variável aleatória sobre Ω cujos valores são todos não-negativos, então*

$$\mathbf{Pr}[X{\geq}\lambda] \;\leq\; \tfrac{1}{\lambda}\, \mathbf{E}[X]$$

para todo número positivo λ.

Demonstração: Se $\{\lambda_1, \ldots, \lambda_n\}$ é o conjunto de valores de X, então $\mathbf{E}[X] = \sum_i \lambda_i \,\mathbf{Pr}[X{=}\lambda_i] \geq \sum_{\lambda_i \geq \lambda} \lambda \,\mathbf{Pr}[X{=}\lambda_i] = \lambda \,\mathbf{Pr}[X{\geq}\lambda]$. □

Às vezes é útil reescrever a desigualdade de Markov assim: $\mathbf{Pr}[X{\geq}\lambda\,\mathbf{E}[X]] \leq \frac{1}{\lambda}$.

Espaço produto

Ω^n Sejam $\mathcal{P}_1, \ldots, \mathcal{P}_n$ medidas discretas de probabilidade sobre Ω. Como é usual, denotamos por Ω^n o conjunto de todas as n-uplas $(\omega_1, \ldots, \omega_n)$ de elementos de Ω. Seja $\mathcal{P} : \Omega^n \to [0,1]$ a função definida por $\mathcal{P}((\omega_1, \ldots, \omega_n)) := \prod_{i=1}^n \mathcal{P}_i(\omega_i)$. Tal $\mathcal{P}$ é uma medida discreta de probabilidade sobre Ω^n. O espaço discreto de probabilidade $(\Omega^n, \mathcal{P})$ é chamado de **espaço produto** e denotado por $(\Omega, \mathcal{P}_1) \times \cdots \times (\Omega, \mathcal{P}_n)$.

Espaços contínuos de probabilidade

A generalização da definição de medida de probabilidade para os chamados espaços contínuos é intuitiva. Infelizmente, sua formalização depende de alguns conceitos não-triviais. Vale ressaltar que espaços contínuos de probabilidade são usados apenas no capítulo 7.

Uma coleção $\mathcal{F}$ de subconjuntos de um conjunto Ω é uma σ-**álgebra** de Ω se $\emptyset \in \mathcal{F}$, $\Omega \setminus A \in \mathcal{F}$ para todo A em $\mathcal{F}$ e $\bigcup_i A_i \in \mathcal{F}$ para toda família $A_1, A_2, \ldots$ de conjuntos de $\mathcal{F}$. Para uma coleção $\mathcal{B}$ de subconjuntos de um conjunto Ω, a σ-**álgebra gerada por** $\mathcal{B}$ é a intersecção de todas as σ-álgebras que contêm $\mathcal{B}$. (Veja os exercícios D.1 e D.2.)

Seja $\mathcal{F}$ uma σ-álgebra de Ω. Uma função $\mathcal{P} : \mathcal{F} \to [0,1]$ é $\mathcal{F}$-**aditiva** se, para toda família $A_1, A_2, \ldots$ de conjuntos de $\mathcal{F}$,

$$\mathcal{P}(\textstyle\bigcup_i A_i) = \sum_i \mathcal{P}(A_i) \,.$$

Dizemos que $\mathcal{P}$ é uma **medida contínua de probabilidade** sobre $(\Omega, \mathcal{F})$ se $\mathcal{P}$ é $\mathcal{F}$-aditiva e $\mathcal{P}(\Omega) = 1$.

Um **espaço contínuo de probabilidade** é um terno $(\Omega, \mathcal{F}, \mathcal{P})$ onde $\mathcal{F}$ é uma σ-álgebra de um conjunto Ω e $\mathcal{P}$ é uma medida contínua de probabilidade sobre $(\Omega, \mathcal{F})$. É fácil concluir do contexto se estamos tratando de medidas ou espaços contínuos ou discretos; assim, o adjetivo pode ser omitido.

Os conceitos definidos para espaços discretos de probabilidade (evento, variável aleatória, esperança, etc.) se estendem naturalmente a espaços contínuos de probabilidade.

Nos próximos exemplos, f_A denota a função característica de um subconjunto A de Ω, ou seja, a função tal que $f_A(\omega) = 1$ se $\omega \in A$ e $f_A(\omega) = 0$ se $\omega \notin A$.

Exemplo D.2: Seja Ω o intervalo $[0, 1)$ e $\mathcal{F}$ a σ-álgebra gerada pela coleção de todos os intervalos da forma $[a, b)$ com $0 \le a < b < 1$. Se, para todo conjunto A em $\mathcal{F}$,

$$\mathcal{P}(A) := \int_\Omega f_A(x)\, dx \,,$$

então $(\Omega, \mathcal{F}, \mathcal{P})$ é um espaço de probabilidade.[1]

Exemplo D.3: Seja Ω a esfera unitária em $\mathbb{R}^n$ centrada na origem, isto é, o conjunto $\{x \in \mathbb{R}^n : \|x\|{=}1\}$. Para quaisquer y e z em Ω, considere a "fatia" $F(y, z) := \{x \in \Omega : xy > 0 \text{ e } \ xz \le 0\}$ da esfera. Seja $\mathcal{F}$ a σ-álgebra gerada pelo conjunto $\{F(y, z) : y, z \in \Omega\}$. A área $\alpha_n := \int_\Omega f_\Omega(x)\, dx$ de Ω depende da paridade de n:

$$\alpha_{2k} = \frac{2\pi^k}{(k-1)!} \quad \text{e} \quad \alpha_{2k+1} = \frac{2^{k+1}\pi^k}{(2k-1)!!} \,,$$

onde $(2k - 1)!! := (2k - 1)(2k - 3)(2k - 5) \cdots (3)(1)$. Se

$$\mathcal{P}(A) := \frac{1}{\alpha_n} \int_\Omega f_A(x)\, dx$$

para todo conjunto A em $\mathcal{F}$, então $(\Omega, \mathcal{F}, \mathcal{P})$ é um espaço de probabilidade.[1]

Distribuição uniforme contínua

Aqui vamos nos restringir a subconjuntos de $\mathbb{R}^n$, pois isso é suficiente para os casos tratados no texto e simplifica a definição.

[1] A função $\mathcal{P} : 2^\Omega \to [0, 1]$, definida de maneira similar sobre a coleção 2^Ω de todos os subconjuntos de Ω, não é 2^Ω-aditiva. Na verdade, não existe função $\mathcal{P} : 2^\Omega \to [0, 1]$ que seja 2^Ω-aditiva. Por isso, precisamos de σ-álgebras.

Se Ω é um subconjunto de $\mathbb{R}^n$ e $\mathcal{F}$ é uma σ-álgebra de Ω, chamamos de **distribuição uniforme contínua** (ou simplesmente distribuição uniforme) qualquer medida de probabilidade contínua $\mathcal{P}$ sobre $(\Omega, \mathcal{F})$ definida por

$$\mathcal{P}(A) := \frac{\int_\Omega f_A(x)\,dx}{\int_\Omega f_\Omega(x)\,dx}$$

para todo conjunto A em $\mathcal{F}$. As medidas de probabilidade apresentadas nos exemplos D.2 e D.3 são exemplos de distribuições uniformes.

Distribuição normal

A **distribuição normal** sobre $\mathbb{R}$ é definida por

$$\mathcal{P}(A) := \frac{1}{\sqrt{2\pi}} \int_\Omega e^{-x^2/2}\,dx$$

para todo conjunto A na σ-álgebra usual de $\mathbb{R}$, que é definida de maneira análoga à do exemplo D.2.

Exercícios

D.1 A coleção de todos os subconjuntos de um conjunto Ω é uma σ-álgebra de Ω.

D.2 Prove que a intersecção de duas σ-álgebras sobre um conjunto Ω é uma σ-álgebra. Prove que a intersecção de infinitas σ-álgebras sobre um conjunto Ω é uma σ-álgebra.

D.3 Se u e v são variáveis aleatórias independentes com distribuição uniforme no intervalo $[0, 1)$, então $x := \cos(2\pi v)\sqrt{-2\ln u}$ é uma variável aleatória com distribuição normal.

D.4 Se $x_1, \ldots, x_n$ são variáveis aleatórias independentes com distribuição normal, então o vetor s com componentes $s_1, \ldots, s_n$ definidas por

$$s_i := \frac{x_i}{\sqrt{x_1^2 + \cdots + x_n^2}}$$

tem distribuição uniforme na esfera unitária.

Apêndice E

Complexidade Computacional

Este apêndice define a terminologia e os fatos básicos de complexidade computacional usados no texto. Um tratamento completo do assunto pode ser encontrado, por exemplo, nos livros de Aho, Hopcroft e Ullman [AHU74], Cormen, Leiserson e Rivest [CLR92], Garey e Johnson [GJ79], Knuth [Knu68], Papadimitriou [Pap94] e Papadimitriou e Steiglitz [PS82].

Palavras

Para resolver um problema usando um computador é necessário descrever os dados do problema por meio de uma seqüência de caracteres. Qualquer seqüência de caracteres será chamada de uma **palavra**.

Não é difícil representar números inteiros, números racionais, vetores, matrizes, grafos, etc. por meio de palavras. Por exemplo, o grafo com $\{a, b, c, d\}$ como conjunto de vértices e $\{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{c, d\}\}$ como conjunto de arestas pode ser representado pela palavra

`({a,b,c,d},{{a,b},{a,c},{a,d},{b,c},{c,d}})` .

Um vetor racional c indexado por um conjunto finito E pode ser representado por uma seqüência de pares (e, c_e), onde e é um elemento de E. Por exemplo, a palavra

`(({a,b},-2),({a,c},12),({a,d},0),({b,c},3/2),({c,d},7))`

codifica um vetor indexado pelo conjunto das arestas do grafo acima. O **tamanho de uma palavra** w, denotado por $\langle w \rangle$, é o número de $\langle w \rangle$

caracteres usados em w (contando-se multiplicidades). Essa definição é um caso concreto do conceito geral de tamanho a que se refere a seção 1.2. Em geral, não fazemos distinção entre um objeto e uma palavra que o representa. Para os nossos propósitos, não há mal em subestimar o tamanho de um objeto.[1] Assim, não é necessário contar rigorosamente os caracteres '`{`', '`}`', '`(`', '`)`' e '`,`' dos exemplos anteriores. O tamanho de um vetor reduz-se então à soma dos tamanhos de seus componentes, ou seja, $\langle c\rangle = \sum_e \langle c_e\rangle$.

O tamanho dos números inteiros e racionais merece especial atenção. O tamanho de um inteiro α é essencialmente[2] $\log|\alpha|$. Analogamente, se α é um número inteiro e β é um número inteiro não nulo então o tamanho do racional α/β é, essencialmente, $\log|\alpha| + \log|\beta|$.

Problemas

Informalmente, um **problema** é uma questão ou tarefa. Exemplos
PCH de problemas são: o **problema do circuito hamiltoniano** (*Hamiltonian circuit problem*), denotado pela sigla PCH e o **problema da árvore**
MST **geradora mínima** (*minimum spanning tree problem*), denotado pela sigla MST.

> **Problema** PCH (G): *Um dado grafo G possui um circuito hamiltoniano?*

> **Problema** MST (G, c): *Dados um grafo G e um custo c_e em $\mathbb{Q}_\geq$ para cada aresta e, encontrar uma árvore geradora de custo mínimo.*

Cada conjunto específico de dados de um problema define uma **instância** do problema. Assim, qualquer grafo define uma instância do problema PCH e qualquer grafo com custos associados às suas arestas é uma instância do problema MST. O **tamanho de uma instância** é o tamanho de uma palavra que representa a instância.

Um problema que pede uma resposta do tipo SIM ou NÃO é chamado de **problema de decisão**. Já um problema que procura um elemento de um conjunto de soluções viáveis que seja melhor possível em relação

---

[1] Como veremos adiante, nossas estimativas serão feitas "a menos de fatores constantes" e portanto não fará diferença dizer que o tamanho de uma palavra é, digamos, metade do número de caracteres.

[2] Mais precisamente, o tamanho de α é $\lceil \log(|\alpha| + 1)\rceil$.

a algum critério é um **problema de otimização**. Os problemas PCH e MST são exemplos de problemas de decisão e otimização, respectivamente.

Algoritmos e modelo de computação

Um **algoritmo**[3] é uma seqüência finita de instruções ou operações que resolve um problema.

Um **modelo de computação** é uma descrição abstrata e conceitual (não necessariamente realista) de um computador que será usado para executar um algoritmo. Um modelo de computação especifica as **operações elementares** que um algoritmo pode executar e o critério empregado para medir a quantidade de tempo que cada operação consome. Exemplo de operações elementares típicas são operações aritméticas entre números e comparações.

O modelo de computação RAM (*random access machine*) [CR73] baseia-se na manipulação de caracteres. Esse modelo é capaz de executar operações envolvendo números racionais. No **critério logarítmico** o consumo de tempo de cada operação depende do tamanho dos operandos (*logarithmic cost criterion*). Por exemplo, a multiplicação de dois números α e β consome essencialmente $\langle\alpha\rangle\,\langle\beta\rangle$ unidades de tempo. RAM

Um outro critério para medir o consumo de tempo, aparentemente menos realista, é o **critério uniforme**, que supõe que cada operação elementar consome uma quantidade de tempo constante (*uniform cost criterion*), ou seja, o consumo de tempo de cada operação elementar independe do tamanho dos operandos. Este modelo é equivalente ao anterior se o tamanho dos operandos for limitado por uma função polinomial do tamanho da instância do problema. Este critério de consumo de tempo constante por operação elementar, juntamente com operandos de tamanho limitado, é empregado em todo o texto, exceto no capítulo 7.

Finalmente, é preciso mencionar um modelo de computação que é útil, embora não seja realista e fuja de nossa convenção sobre representação de objetos por palavras. Esse modelo, usado no capítulo 7, é capaz de manipular números reais arbitrários. Supõe-se que cada operação envolvendo números reais consome apenas uma unidade de tempo, mesmo uma operação como raiz quadrada. Este modelo é chamado de real-RAM [PS85]. real-RAM

[3] A formalização matemática clássica de um algoritmo é a máquina de Turing.

Análise de algoritmos e algoritmos polinomiais

Ao analisarmos um algoritmo em um determinado modelo de computação queremos estimar o seu consumo de tempo[4] como uma função do tamanho da instância do problema. Por exemplo, exprimimos o consumo de tempo de algoritmos para o $\mathrm{MST}(G, c)$ como uma função de $\langle (G, c) \rangle$.

Ao expressar a quantidade de tempo consumida por um algoritmo, ignoramos os fatores constantes. Mais precisamente, dizemos que um algoritmo A resolve um problema em tempo $O(f(n))$, para uma função f de $\mathbb{Z}_{\geq}$ em $\mathbb{Z}_{\geq}$, se existe uma constante c tal que a quantidade de tempo consumida[5] por A para resolver uma instância I do problema é limitada por $c\, f(\langle I \rangle)$. Isto corresponde à chamada **análise do pior caso** (*worst-case analysis*) do algoritmo. O algoritmo A é **polinomial** se resolve o problema em tempo $O(p(n))$ para alguma função polinomial p. É claro que o conceito de algoritmo polinomial depende não só do algoritmo mas também do modelo de computação.

$O(f(n))$

Critério de eficiência: classes P, NP e co-NP

Por um **algoritmo eficiente** entendemos um algoritmo polinomial. Este critério de eficiência foi proposto, independentemente, por Cobham [Cob65] e Edmonds [Edm65b]. A classe de todos os problemas de decisão que podem ser resolvidos por algoritmos polinomiais é denotada por P. Por exemplo, a versão de decisão do problema[6] $\mathrm{MST}(G, c)$ está em P, já que existem algoritmos polinomiais para resolvê-la [CLR92].

P

NP

A classe NP (abreviação de *nondeterministic polynomial time*) é composta pelos problemas de decisão para os quais uma resposta afirmativa possui um certificado que pode ser verificado em tempo polinomial. Mais precisamente, a classe NP é formada pelos problemas de decisão Π para os quais existe um problema Π' em P e uma função polinomial $p(n)$ tais que, para cada instância I do problema Π, existe um objeto C com $\langle C \rangle \leq p(\langle I \rangle)$ tal que a resposta a $\Pi(I)$ é SIM se e somente se a resposta a $\Pi(I, C)$ é SIM. O objeto C é dito um **certificado polinomial** (ou

[4] A quantidade de espaço consumida pelo algoritmo é negligenciada no presente texto.

[5] Está incluído aqui, implicitamente, o tempo necessário para que o algoritmo decida se a palavra dada é uma representação válida de uma instância do problema.

[6] Dados um grafo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e de G e um número α em $\mathbb{Q}_{\geq}$, existe uma árvore geradora em G de custo não superior a α?

certificado curto) da resposta SIM a $\Pi(I)$. Com um algoritmo polinomial para Π', um verificador incrédulo pode, após gastar uma quantidade de tempo polinomial, testar a validade de uma resposta SIM. Por exemplo, se a resposta a uma dada instância G do problema do circuito hamiltoniano é SIM então um circuito hamiltoniano em G é um objeto que certifica a resposta: dados um grafo G e um circuito H de G pode-se verificar em tempo $O(\langle G\rangle)$ se H é um circuito hamiltoniano. Logo, o problema PCH, bem como "versões de decisão" de todos os problemas no texto, pertence a NP.

Claramente $\mathrm{P} \subseteq \mathrm{NP}$, já que se Π é um problema em P, então pode-se tomar a seqüência de instruções realizadas por um algoritmo polinomial para resolver $\Pi(I)$ como certificado polinomial da resposta SIM a $\Pi(I)$. É crença de muitos que a classe NP é maior que a classe P, ainda que isso não tenha sido provado até agora.[7] Este é o intrigante problema matemático conhecido pelo rótulo "$\mathrm{P} \neq \mathrm{NP}$?". $\mathrm{P} \neq \mathrm{NP}$?

A classe co-NP é definida trocando-se SIM por NÃO na definição de NP, ou seja, um problema de decisão Π está em co-NP se admite um certificado polinomial para a resposta NÃO. Os problemas em $\mathrm{NP} \cap$ co-NP admitem certificados polinomiais para as respostas SIM e NÃO. Em particular, é evidente que $\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$. co-NP

NP-completude

Uma **redução**[8] de um problema Π a um problema Π' é um algoritmo A que resolve Π usando uma subrotina hipotética A' que resolve Π', de tal forma que, se A' é um algoritmo polinomial, então A é um algoritmo polinomial. Usamos a notação $\Pi \leq_{\mathrm{T}} \Pi'$ para denotar a existência de uma redução de Π a Π'. Dizemos que um problema Π pode ser reduzido a um problema Π' se $\Pi \leq_{\mathrm{T}} \Pi'$. Dois problemas são ditos **polinomialmente equivalentes** se um pode ser reduzido ao outro. $\leq_{\mathrm{T}}$

Um problema Π em NP é **NP-completo** se cada problema em NP pode ser reduzido a Π. É evidente que se Π' pertence a P e Π pode ser reduzido a Π', então Π pertence a P. Isto implica que existe um algoritmo polinomial para um problema NP-completo se e somente se $\mathrm{P} = \mathrm{NP}$.

O primeiro problema que se demonstrou ser NP-completo é o **problema da satisfatibilidade** (*satisfiability problem*). Este problema é

[7] Para uma introdução às classes P e NP baseada na fábula dos Cavaleiros da Távola Redonda, veja Lovász e Plummer [LP86] e Kohayakawa e Soares [KS95].

[8] Conhecida como redução de Turing ou redução de Cook [GJ79].

uma simplificação do problema MAXSAT descrito no capítulo 6.

Problema SAT ($\mathcal{C}$): *Dada uma coleção $\mathcal{C}$ de cláusulas booleanas, existe uma valoração que satisfaz todas as cláusulas em $\mathcal{C}$?*

Teorema E.1 (Cook [Coo71], Levin [Lev73]): *O problema* SAT *é* NP*-completo.* □

Um problema Π, não necessariamente em NP, é **NP-díficil** se a existência de um algoritmo polinomial para Π implica em P = NP. Assim, por exemplo, MAXSAT é NP-difícil, já que um algoritmo polinomial para MAXSAT pode ser adaptado para resolver SAT em tempo polinomial; como SAT é NP-completo, a existência de um tal algoritmo implica em P = NP. Cook [Coo71] e Karp [Kar72] mostraram que vários problemas discutidos no presente texto são NP-difíceis.

Teorema E.2: *Os problemas* ESCALONAMENTO, EMPACOTAMENTO, MAXCUT, MAXSAT, MINCC, MINCV, MINFS, MINMCUT, MINTC, MOCHILA, TSPM *e* TSP *são* NP-*difíceis.* □

Algoritmos fortemente polinomiais

Um algoritmo polinomial é **fortemente polinomial** ou **genuinamente polinomial** se o número de operações elementares realizadas não depende do tamanho dos números (caso exista algum) que definem uma instância do problema. O algoritmo de Kruskal [CLR92] para o problema $\mathrm{MST}(G, c)$ é um exemplo de algoritmo fortemente polinomial, já que o número de operações elementares realizadas é proporcional a $\langle G\rangle \log\langle G\rangle$. Vários algoritmos mencionados neste texto são fortemente polinomiais. Um exemplo de algoritmo polinomial que não é fortemente polinomial é o algoritmo de Euclides [Sch86] para encontrar o máximo divisor comum entre dois inteiros positivos α e β, já que o número de operações elementares realizadas pelo algoritmo é essencialmente $\langle\alpha\rangle + \langle\beta\rangle$. Outro exemplo não fortemente polinomial é o algoritmo dos elipsóides de Khachiyan [Kha79] que resolve problemas de programação linear e realiza um número de operações elementares que depende do tamanho dos números envolvidos (apêndice C).

Algoritmos pseudopolinomiais

Seja Π um problema e $\mathcal{I}$ o seu conjunto de instâncias. Para cada I em $\mathcal{I}$ seja $\mathrm{Max}(I)$ o maior número inteiro em valor absoluto que ocorre em I; defina $\mathrm{Max}(I) := 0$ se nenhum número inteiro ocorre em I. Aqui estamos supondo que cada número racional é representados como quociente entre dois números inteiros. Por exemplo, $\mathrm{Max}(3/17) = 17$. Um **algoritmo pseudopolinomial** para Π é um algoritmo que consome uma quantidade de tempo limitada por uma função polinomial nas variáveis $\langle I\rangle$ e $\mathrm{Max}(I)$. É claro que todo algoritmo polinomial também é pseudopolinomial. O algoritmo MOCHILA-EXATO do capítulo 2 é um algoritmo pseudopolinomial que não é polinomial. $\mathrm{Max}(I)$

Seja $\mathcal{I}_f := \{I \in \mathcal{I} : \mathrm{Max}(I) \le f(\langle I\rangle)\}$ para alguma função f de $\mathbb{Z}_\geq$ em $\mathbb{Z}_\geq$ e seja Π_f o problema Π restrito ao conjunto de instâncias $\mathcal{I}_f$. O problema Π é **NP-completo no sentido forte** ou **fortemente NP-completo** se Π_p é NP-completo para alguma função polinomial p. Em particular, problemas como SAT, que não envolvem números, são NP-completos no sentido forte. Considere, por exemplo, a versão a seguir do problema do caixeiro viajante (TSP) como problema de decisão.

> **Problema** $\mathrm{TSP_D}\,(G, c, \beta)$: *Dados um grafo G, um custo c_e em $\mathbb{Q}_\geq$ para cada aresta e de G e um número racional β, existe um circuito hamiltoniano em G de custo não superior a β?*

O problema $\mathrm{TSP_D}$ é NP-completo no sentido forte, já que o problema PCH é essencialmente esta versão de decisão do TSP restrita a instâncias onde o custo c_e de cada aresta e é 1 ou $|V_G|+1$ e β é $|V_G|$ (demonstração do teorema 8.5).

Um problema Π, não necessariamente em NP, é **NP-difícil no sentido forte** se a existência de um algoritmo pseudopolinomial para Π implica em P = NP. O TSP é um exemplo de problema NP-difícil no sentido forte. Todos os problemas no enunciado do teorema E.2, com exceção do problema MOCHILA, são NP-difíceis no sentido forte.

> **Teorema E.3**: *Os problemas* ESCALONAMENTO, EMPACOTAMENTO, MAXCUT, MAXSAT, MINCC, MINCV, MINFS, MINMCUT, MINTC, TSPM *e* TSP *são NP-difíceis no sentido forte.* □

Mais sobre algoritmos pseudopolinomiais e NP-completude no sentido forte pode ser encontrado em Garey e Johnson [GJ79] e em Papadimitriou e Steiglitz [PS82].

Exercício

E.1 Mostre que se $\Pi_1 \leq_{\mathrm{T}} \Pi_2$ e $\Pi_2 \leq_{\mathrm{T}} \Pi_3$, então $\Pi_1 \leq_{\mathrm{T}} \Pi_3$.

E.2 Mostre que se Π é um problema de decisão, Π' está em P e $\Pi \leq_{\mathrm{T}} \Pi'$, então Π está em P.

E.3 Mostre que se Π é um problema de decisão, Π' está em NP e $\Pi \leq_{\mathrm{T}} \Pi'$, então Π está em NP.

E.4 Mostre que se Π' é NP-completo e $\Pi' \leq_{\mathrm{T}} \Pi$, então Π é NP-difícil.

E.5 Mostre que o problema $\mathrm{TSP}_{\mathrm{D}}$ está em NP e que os problemas TSP e $\mathrm{TSP}_{\mathrm{D}}$ são polinomialmente equivalentes.

Bibliografia

[ABCC98] D. Applegate, R. Bixby, V. Chvátal e W. Cook. On the solution of traveling salesman problems. *Documenta Mathematica*, Vol. Extra III:645–656, 1998. Proceedings of the International Congress of Mathematicians. URL: `http://www.mathematik.uni-bielefeld.de/documenta/xvol-icm/17/17.html`. [p. 21]

[ABI86] N. Alon, L. Babai e A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986. [p. 74]

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela e M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999. [p. iii, 2, 93, 97, 99]

[ACP95] G. Ausiello, P. Crescenzi e M. Protasi. Approximate solution of NP optimization problems. *Theoretical Computer Science*, 150(1):1–55, 1995. [p. 98, 99]

[ADP80] G. Ausiello, A. D'Atri e M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21:136–153, 1980. [p. 96]

[AG94] M. Aggarwal e N. Garg. A scaling technique for better network design. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 233–240, 1994. [p. 60]

[AGK+98] S. Arora, M. Grigni, D.R. Karger, P.N. Klein e A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the Nineth ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 33–41, 1998. [p. 22]

[AHU74] A.V. Aho, J.E. Hopcroft e J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. [p. 15, 125]

[AK98] N. Alon e N. Kahale. Approximating the independence number via the θ-function. *Mathematical Programming*, 80:231–237, 1998. [p. 85]

[AKR95] A. Agrawal, P. Klein e R. Ravi. When trees collide: an approximation algorithm for the generalizad Steiner problem on networks. *SIAM Journal on Computing*, 24(2):440–456, 1995. [p. 60]

[Ali95] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5:13–51, 1995. [p. 85]

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan e M. Szegedy. Proof verification and intractability of approximation problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, p. 24–27, 1992. [p. 2, 99]

[AMO93] R.K. Ahuja, T.L. Magnanti e J.B. Orlin. *Network Flows*. Prentice-Hall, 1993. [p. 26, 27, 33]

[AR98] Y. Aumann e Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998. [p. 33]

[Aro94] S. Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems*. PhD Thesis, Princeton University, 1994. [p. 2]

[Aro95] S. Arora. Reductions, codes, PCPs, and inapproximability. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, p. 404–413, 1995. [p. 2]

[Aro98] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998. [p. 22]

[AS92] N. Alon e J. Spencer. *The Probabilistic Method*. John Wiley, 1992. [p. 70]

[AW00] T. Asano e D.P. Williamson. Improved approximation algorithms for MAX SAT. In *Proceedings of the Eleventh ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 96–115, 2000. [p. 74]

[BKRV98] A. Blum, G. Konjevod, R. Ravi e S. Vempala. Semidefinite relaxations for minimum bandwidth and other vertex-ordering problems. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, p. 100–105, 1998. [p. 85]

[Blu91] A. Blum. *Algorithms for Approximate Graph Coloring*. PhD Thesis, Massachusetts Institute of Technology, 1991. [p. 2]

[BM76] J.A. Bondy e U.S.R. Murty. *Graph Theory with Applications*. Macmillan/Elsevier, 1976. [p. 15, 101, 104]

[Bru98] P. Brucker. *Scheduling Algorithms*. Springer, 2. ed., 1998. [p. 21]

[BTV99] D. Bertsimas, C. Teo e R. Vohra. On dependent randomized rounding algorithms. *Operations Research Letters*, 24(3):105–114, 1999. [p. 74]

[BYE81] R. Bar-Yehuda e S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981. [p. 47, 60]

[CFR98] G. Călinescu, C.G. Fernandes e B. Reed. Multicuts in unweighted graphs with bounded degree and bounded treewidth. In R.E. Bixby, E.A. Boyd e R.Z. Ríos-Mercado, editores, *Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 1412 of *Lecture Notes in Computer Science*, p. 137–152. Springer, 1998. [p. 33]

[CG89] B. Chor e O. Goldreich. On the power of two-point sampling. *Journal of Complexity*, 5:96–106, 1989. [p. 75]

[Chr76] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Carnegie Mellon University, 1976. [p. 17, 21]

[Chv79] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. [p. 8, 39]

[Chv83] V. Chvátal. *Linear Programming*. Freeman, 1983. [p. 113]

[CJLL95] P. Chrétienne, E.G. Coffman Jr., J.K. Lenstra e Z. Liu, editores. *Scheduling Theory and its Applications*. John Wiley & Sons, 1995. [p. 21]

[CK00] P. Crescenzi e V. Kann. *A Compendium of NP Optimization Problems*, 2000. URL: `http://www.nada.kth.se/~viggo/wwwcompendium/`. [p. 99]

[CKR00] G. Călinescu, H. Karloff e Y. Rabani. An improved approximation algorithm for MULTIWAY CUT. *Journal of Computer and System Sciences*, 60(3):564–574, 2000. [p. 33, 74]

[CLR92] T.H. Cormen, C.E. Leiserson e R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1992. [p. 10, 15, 20, 59, 125, 128, 130]

[Cob65] A. Cobham. The intrinsic computational difficult of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science*, p. 24–30. North-Holland, 1965. [p. 128]

[Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC)*, p. 151–158, 1971. [p. 87, 130]

[CR73] S.A. Cook e R.A. Reckhow. Time-bounded random access machines. *Journal of Computer and System Science*, 7:354–375, 1973. [p. 127]

[CRW01] F. Chudak, T. Roughgarden e D.P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. In *Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization Conference (IPCO)*, 2001. Aceito para publicação. [p. 61, 85]

[CS98] B. Chor e M. Sudan. A geometric approach to betweenness. *SIAM Journal on Discrete Mathematics*, 11(4):511–523, 1998. [p. 85]

[Dev86] L. Devroye. *Non-uniform Random Variate Generation*. Springer, 1986. [p. 75, 84]

[Die00] R. Diestel. *Graph Theory*. Springer, 2. ed., 2000. [p. 101]

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, p. 269–271, 1959. [p. 59]

[DJP+94] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour e M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994. [p. 26, 33, 34, 99]

[DL97] M.M. Deza e M. Laurent. *Geometry of Cuts and Metrics*. Springer, 1997. [p. 85]

[Edm65a] J. Edmonds. Maximum matching and a polyhedron with $0, 1$-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965. [p. 22, 60]

[Edm65b] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. [p. 22, 128]

[EK00] L. Engebretsen e M. Karpinski. Approximation hardness of TSP with bounded metrics. *Eletronic Colloquium on Computational Complexity*, 7(89), 2000. URL: `http://www.eccc.uni-trier.de/eccc/`. [p. 96]

[ENRS95] G. Even, J. Naor, S. Rao e B. Schieber. Divide-and-conquer approximation algorithms via spreading metrics. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, p. 62–71, 1995. [p. 33]

[Erd67] P. Erdős. Gráfok páros körüljárású részgráfjairól (On bipartite subgraphs of graphs, em húngaro). *Matematikai Lapok*, 18:283–288, 1967. [p. 2, 85]

[ES73] P. Erdős e J. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Ser. A*, 14:298–301, 1973. [p. 74]

[ES74] P. Erdős e J. Spencer. *The Probabilistic Method in Combinatorics.* Academic Press, 1974. [p. 70]

[Fei99] U. Feige. Randomized rounding for semidefinite programs—variations on the MAX CUT example. In *Randomization, Approximation, and Combinatorial Optimization*, volume 1671 of *Lecture Notes in Computer Science*, p. 189–196. Springer, 1999. [p. 74]

[Feo01] P. Feofiloff. *Algoritmos de Programação Linear.* A ser publicado pela EDUSP, 2001. [p. 113]

[FF56] L.R. Ford e D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. [p. 33, 60]

[FG95] U. Feige e M.X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems (ISTCS)*, p. 182–189, 1995. [p. 74, 85]

[FJ97] A. Frieze e M. Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica*, p. 67–81, 1997. [p. 85]

[GGL95] R.L. Graham, M. Grötschel e L. Lovász, editores. *Handbook of combinatorics. Vol. 1, 2.* Elsevier Science B.V., Amsterdam, 1995. [p. 85]

[GGP+94] M.X. Goemans, A.V. Goldberg, S. Plotkin, D.B. Shmoys, É. Tardos e D.P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the Fifth ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 223–232, 1994. [p. 60]

[GGU72] M.R. Garey, R.L. Graham e J.D. Ullman. Worst-case analysis of memory allocation algorithms. In *Proceedings of the 4th Annual ACM Symposium on the Theory of Computing (STOC)*, p. 143–150, 1972. [p. 2]

[GGW98] H.N. Gabow, M.X. Goemans e D.P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Mathematical Programming*, 82(1-2, Ser. B):13–40, 1998. [p. 60, 61]

[GJ75] M.R. Garey e D.S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4:397–411, 1975. [p. 98]

[GJ78] M.R. Garey e D.S. Johnson. Strong NP-completeness results: Motivations, examples and implications. *Journal of the ACM*, 25:499–508, 1978. [p. 91]

[GJ79] M.R. Garey e D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, 1979. [p. 6, 8, 12, 14, 19, 20, 35, 49, 65, 77, 96, 98, 125, 129, 131]

[GLS93] M. Grötschel, L. Lovász e A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 2. ed., 1993. [p. 82, 84, 117, 118]

[Goe97] M.X. Goemans. Semidefinite programming in combinatorial optimization. *Mathematical Programming*, 79:143–161, 1997. [p. 84]

[Gon89] C.C. Gonzaga. An algorithm for solving linear programming problems in $O(n^3L)$ operations. In N. Megiddo, editor, *Progress in Mathematical Programming: Interior Point and Related Methods*, p. 1–28. Springer, 1989. [p. 117]

[Gon92] C.C. Gonzaga. Path following methods for linear programming. *SIAM Review*, 34(2):167–227, 1992. [p. 117]

[Gra66] R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966. [p. 2, 6]

[Gra69] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969. [p. 21]

[Gui98] K.S. Guimarães. Algoritmos de aproximação para problemas de otimização. In H.P. de Moura, editor, *XVII Jornada de Atualização em Informática*, volume II, p. 1–47. SBC, 1998. [p. v, 20]

[GVY96] N. Garg, V.V. Vazirani e M. Yannakakis. Approximate max-flow and min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25:235–251, 1996. [p. 26, 33]

[GVY97] N. Garg, V.V. Vazirani e M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997. [p. 33, 59, 61]

[GW94] M.X. Goemans e D.P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666, 1994. [p. 67, 69, 74]

[GW95a] M.X. Goemans e D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995. [p. 50, 52, 59, 60]

[GW95b] M.X. Goemans e D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995. [p. 79, 85]

[GW98] M.X. Goemans e D.P. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica*, 18(1):37–59, 1998. [p. 61]

[GW01] M.X. Goemans e D.P. Williamson. Approximation algorithms for MAX 3-CUT and other problems via complex semidefinite programming. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing (STOC)*, 2001. Aceito para publicação. [p. 85]

[Hal00] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. In *Proceedings of the Eleventh ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 329–337, 2000. [p. 85]

[Hås97] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC)*, p. 1–10, 1997. [p. 96]

[HH86] N.G. Hall e D.S. Hochbaum. A fast approximation algorithm for the multicovering problem. *Discrete Applied Mathematics*, 15(1):35–40, 1986. [p. 38]

[Hoc82] D.S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982. [p. 24, 36, 38, 60]

[Hoc97] D.S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, 1997. [p. iii, 2, 20, 33, 38, 59, 60, 74]

[Hoo91] J.A. Hoogeveen. Analysis of Christofides' heuristic: some paths are more difficult than cycles. *Operations Research Letters*, 10:291–295, 1991. [p. 22]

[HPS94] S. Hougardy, H.-J. Prömel e A. Steger. Probabilistically checkable proofs and their consequences for approximation algorithms. *Discrete Mathematics*, 136:175–223, 1994. [p. 99]

[HS87] D.S. Hochbaum e D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987. [p. 39]

[HS88] D.S. Hochbaum e D.B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approach. *SIAM Journal on Computing*, 17:539–551, 1988. [p. 21, 39, 89, 96]

[Hu63] T.C. Hu. Multicommodity network flows. *Operations Research*, 9:898–900, 1963. [p. 26, 33]

[IK75] O.H. Ibarra e C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22:463–468, 1975. [p. 12, 96]

[Ita78] A. Itai. Two-commodity flow. *Journal of the ACM*, 25(4):596–611, 1978. [p. 26, 33]

[JMVW99] K. Jain, I. Măndoiu, V.V. Vazirani e D.P. Williamson. A primal-dual schema based approximation algorithm for the element connectivity problem. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 484–489, 1999. [p. 61]

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974. [p. 2, 21, 65, 74]

[Joh87] M.E. Johnson. *Multivariate Statistical Simulation.* John Wiley, 1987. [p. 75, 84]

[JRR95] M. Jünger, G. Reinelt e G. Rinaldi. The traveling salesman problem. In M.O. Ball, T.L. Magnanti, C.L. Monma e G.L. Nemhauser, editores, *Network Models*, p. 225–330. North-Holland, 1995. [p. 21]

[JV00] K. Jain e V.V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. *17th International Symposium on Mathematical Programming (ISMP)*, 2000. URL: `http://www.cc.gatech.edu/people/home/kjain/`. [p. 61, 74]

[Kan92] V. Kann. *On the Approximability of NP-complete Optimization Problems.* PhD Thesis, Royal Institute of Technology, Sweden, 1992. [p. 2]

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller e J.M. Thatcher, editores, *Complexity of Computer Computations*, p. 85–103. Plenum, 1972. [p. 87, 90, 91, 130]

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. [p. 117]

[Kar96] H. Karloff. How good is the Goemans-Williamson MAX CUT algorithm? In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC)*, p. 427–434, 1996. [p. 85]

[KG98] J. Kleinberg e M.X. Goemans. The Lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM Journal on Discrete Mathematics*, 11(2):196–204, 1998. [p. 85]

[Kha79] L.G. Khachiyan. A polynomial algorithm for linear programming. *Soviet Mathematical Doklady*, 20:191–194, 1979. [p. 117, 130]

[KKS+99] D.R. Karger, P. Klein, C. Stein, M. Thorup e N.E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, p. 668–678, 1999. [p. 34]

[KMS98] D. Karger, R. Motwani e M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2):246–265, 1998. [p. 85]

[KMSV99] S. Khanna, R. Motwani, M. Sudan e U. Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1):164–191, 1999. [p. 95, 99]

[Knu68] D.E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, 1968. [p. 125]

[Knu98] D.E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 3. ed., 1998. [p. 63, 72, 75, 84]

[KP99] H. Kellerer e U. Pferschy. A new fully polynomial time approximation scheme for the knapsack problem. *Journal of Combinatorial Optimization*, 3(1):59–71, 1999. [p. 21]

[KRAR95] P. Klein, S. Rao, A. Agrawal e R. Ravi. An approximate max-flow min-cut relation for undirected multicommodity flow, with applications. *Combinatorica*, 15(2):187–202, 1995. [p. 33]

[KS95] Y. Kohayakawa e J. Soares. *Demonstrações Transparentes e a Impossibilidade de Aproximações*. Livro do XX Colóquio Brasileiro de Matemática. IMPA, 1995. [p. 99, 129]

[Kuh55] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. [p. 60]

[KZ97] H. Karloff e U. Zwick. A 7/8-approximation for MAX 3-SAT? In *Proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS)*, p. 406–415, 1997. URL: `http://www.math.tau.ac.il/~zwick/`. [p. 74, 85]

[Law76] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976. [p. 38]

[Lev73] L.A. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973. [p. 130]

[LK73] S. Lin e B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973. [p. 21]

[LLR95] N. Linial, E. London e Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. [p. 33]

[LLRS90] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan e D.B. Shmoys, editores. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley, 1990. Reprint of the 1985 original. [p. 21]

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975. [p. 21, 39]

[Lov01] L. Lovász. Semidefinite programs and combinatorial optimization. In *Brazilian Summer School on Combinatorics and Algorithms*, p. 1–58. CIMPA, 2001. A ser publicado pela Springer. [p. 78, 85]

[LP86] L. Lovász e M.D. Plummer. *Matching Theory*. Elsevier, 1986. [p. 17, 22, 129]

[LR99] T. Leighton e S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999. [p. 33]

[Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986. [p. 74]

[LW95] M. Luby e A. Wigderson. Pairwise independence and derandomization. Technical Report TR-035, International Computer Science Institute, 1995. [p. 75]

[Mit99] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999. [p. 22]

[MPS98] E. Mayr, H.J. Prömel e A. Steger, editores. *Lectures on Proof Verification and Approximation Algorithms*, volume 1367 of *Lecture Notes in Computer Science*. Springer, 1998. [p. iii, 2, 99]

[MR95a] S. Mahajan e H. Ramesh. Derandomizing semidefinite programming based approximation algorithms. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, p. 162–169, 1995. [p. 85]

[MR95b] R. Motwani e P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. [p. 74]

[MT90] S. Martello e P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley, 1990. [p. 21]

[OM87] P. Orponen e H. Mannila. On approximation preserving reductions: Complete problems and robust measures. Technical Report C-28, University of Helsinki, 1987. [p. 96]

[Pad99] M. Padberg. *Linear Optimization and Extensions*. Springer, 2. rev. exp. ed., 1999. [p. 113]

[Pap94] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. [p. 98, 125]

[PS82] C.H. Papadimitriou e K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982. [p. 59, 117, 125, 131]

[PS85] F.P. Preparata e M.I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985. [p. 127]

[PV00] C.H. Papadimitriou e S. Vempala. On the approximability of the traveling salesman problem. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, p. 126–133, 2000. [p. 99]

[PY91] C.H. Papadimitriou e M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991. [p. 95, 99]

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988. [p. 74]

[Rav94] R. Ravi. A primal-dual approximation algorithm for the Steiner forest problem. *Information Processing Letters*, 50(4):185–189, 1994. [p. 60]

[Rei00] G. Reinelt. TSPLIB – *A Library of Sample Instances for the* TSP, 2000. URL: `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`. [p. 21]

[RS97] R. Raz e S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, p. 475–484, 1997. [p. 10, 96]

[RSL77] D.J. Rosenkrantz, R.E. Stearns e P.M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6:563–581, 1977. [p. 16]

[RT87] P. Raghavan e C.D. Thompson. Randomized rounding. *Combinatorica*, 7:365–374, 1987. [p. 74]

[RV99] S. Rajagopalan e V.V. Vazirani. Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28(2):526–541, 1999. [p. 33, 61]

[RW95] R. Ravi e D.P. Williamson. An approximation algorithm for minimum-cost vertex-connectivity problems. In *Proceedings of the Sixth ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 332–341, 1995. [p. 60]

[RZ00] G. Robins e A. Zelikovsky. Improved steiner tree approximation in graphs. In *Proceedings of the Tenth ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 770–779, 2000. [p. 60]

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley, 1986. [p. 59, 82, 113, 117, 118, 130]

[SG76] S. Sahni e T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976. [p. 90, 96, 98]

[Shm98] D.B. Shmoys. Using linear programming in the design and analysis of approximation algorithms: two illustrative problems. In K. Jansen e J. Rolin, editores, *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization*, number 1444 in Lecture Notes in Computer Science, p. 15–32. Springer, 1998. [p. 33]

[Sku98] M. Skutella. Semidefinite relaxations for parallel machine scheduling. In *Proceedings of the 39th Symposium on Foundations of Computer Science (FOCS)*, p. 472–481, 1998. [p. 85]

[Sku99] M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling, 1999. [p. 85]

[Spe87] J. Spencer. *Ten Lectures on the Probabilistic Method*. SIAM, 1987. [p. 70, 74]

[Sri99] A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29(2):648–670, 1999. [p. 33]

[Ste01] A. Steger. Approximability of NP-optimization problems. In *Brazilian Summer School on Combinatorics and Algorithms*, p. 59–115. CIMPA, 2001. A ser publicado pela Springer. [p. 98]

[Tre96] L. Trevisan. *Reductions and (Non-)Approximability*. PhD Thesis, Università di Roma "La Sapienza", 1996. [p. 2]

[Vaz01] V.V. Vazirani. *Approximation Algorithms*. A ser publicado pela Springer, 2001. [p. iii, 2, 21, 33, 74, 85]

[WG94] D.P. Williamson e M.X. Goemans. Computational experience with an approximation algorithm on large-scale Euclidean matching instances. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 355–364, 1994. [p. 60]

[WGMV95] D.P. Williamson, M.X. Goemans, M. Mihail e V.V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15:435–454, 1995. [p. 60]

[Wil93] D.P. Williamson. *On the Design of Approximation Algorithms for a Class of Graph Problems*. PhD Thesis, Massachusetts Institute of Technology, 1993. [p. 2]

[Wil98] D.P. Williamson. Lecture notes on approximation algorithms. Technical Report RC 21409, T.J. Watson Research Center (IBM Research Division), 1998. [p. 20, 85]

[WSV00] H. Wolkowicz, R. Saigal e L. Vandenberghe, editores. *Handbook of Semidefinite Programming: Theory, Algorithms, and Applications*. Kluwer, 2000. [p. 86]

[Yan94] M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17(3):475–502, 1994. [p. 74]

[Zwi99] U. Zwick. Outward rotations: a tool for rounding solutions of semidefinite programming relaxations, with applications to MAX CUT and other problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, p. 679–687, 1999. [p. 74, 85]

Índice