Programação Inteira

Flávio K. Miyazawa[1†]

[1]Instituto de Computação — Universidade Estadual de Campinas
Caixa Postal 6176 — 13084–971 — Campinas–SP — Brazil,

fkm@ic.unicamp.br

Resumo. *Neste texto apresentamos uma introdução a algumas técnicas e conceitos básicos envolvidos em programação inteira. Consideramos alguns problemas para os quais é possível obter soluções de forma eficiente e outros considerados na literatura como sendo difíceis. Formulamos estes problemas através de modelos de programação inteira e aplicamos técnicas para a obtenção de soluções a partir destes modelos.*
As técnicas que consideramos são usadas para atacar diversos problemas práticos como projeto de redes de telecomunicações, projeto de circuitos VLSI, corte de materiais, empacotamento em containers, localização de centros distribuidores, escalonamento de tarefas, roteamento de veículos, etc.

Abstract. *In this text, we present a short introduction to some basic techniques and concepts in integer programming. We consider problems solved efficiently and $\mathcal{NP}$-hard problems. In both cases, we formulate the problems by linear and integer programming models and present techniques to obtain exact or reduced cost solutions.*
The techniques we consider are applied to many practical problems, like network design, VLSI circuit design, cutting and packing, facility location, scheduling, vehicle routing problems, etc.

1. Introdução

Neste texto apresentamos uma breve introdução a algumas técnicas e conceitos básicos envolvidos em programação inteira. As técnicas que consideramos fazem parte da área de otimização combinatória e são usadas para atacar diversos problemas práticos.

Os problemas de otimização podem ser de minimização ou de maximização. Em ambos os casos, temos uma função aplicada a um domínio finito, que em geral é enumerável. Nosso objetivo é encontrar um elemento deste domínio de valor ótimo, que pode ser de valor mínimo, caso o problema seja de minimização, ou máximo, caso o problema seja de maximização. Apesar de finito, o domínio da função é em geral grande e algoritmos ingênuos que verificam cada elemento deste domínio se tornam impraticáveis. Desta maneira, surge a necessidade de usar técnicas mais elaboradas para encontrar estas soluções de valor ótimo.

[†]Financiado em parte pelo MCT/CNPq, projeto PRONEx (Proc. 664107/97-4) e pelo CNPq (Proc. 470608/01–3, 464114/00–4, 300301/98–7).

Veremos que alguns problemas de otimização combinatória podem ser bem resolvidos, mas para muitos outros não são esperados algoritmos eficientes para se obter uma solução ótima. Muitos destes problemas são importantes e necessitam de soluções, mesmo que não sejam de valor ótimo (de preferência que tenham valor próximo do ótimo) ou sejam obtidas após muito processamento computacional.

A área de otimização combinatória apresenta uma grande diversidade de problemas práticos. Os problemas que veremos encontram aplicações em projeto de redes de telecomunicações, projeto de circuitos VLSI, corte de materiais, empacotamento em containers, localização de centros distribuidores, escalonamento de tarefas, roteamento de veículos, seqüênciamento de DNA, etc.

Para tratar estes problemas, muitas vezes os modelamos através de uma formulação em programação inteira e posteriormente aplicamos alguma técnica para obter uma solução ótima ou aproximada.

Na próxima seção apresentamos alguns conceitos e definições em Teoria dos Grafos e em Complexidade Computacional. Na seção 3, apresentamos alguns problemas que podem ser resolvidos de forma eficiente através do uso de programação linear. Na seção 4, apresentamos algumas técnicas e modelos para problemas de programação linear inteira e na seção 5, apresentamos algumas técnicas para a obtenção de soluções a partir dos modelos de programação inteira.

Por ser um texto resumido, recomendamos que o leitor interessado em aprender mais sobre esta área, consulte outros livros na literatura, como os listados a seguir [Ferreira and Wakabayashi, 1996, Nemhauser and Wolsey, 1988, Papadimitriou and Steiglitz, 1982, Schrijver, 1986, Wolsey, 1998].

2. Preliminares

Para um melhor entendimento das estruturas e técnicas usadas neste texto, apresentamos nesta seção alguns conceitos básicos em Teoria dos Grafos e Complexidade Computacional.

2.1. Teoria dos Grafos

Dado um conjunto E, uma função $c : E \to \mathbb{R}$ e um subconjunto F de E, vamos denotar por $c(E)$ como a soma dos valores de c aplicado nos elementos de F (i.e., $c(F) := \sum_{e \in F} c(e)$).

Um *grafo não-orientado (resp. orientado)* é um par ordenado (V, E) onde V é um conjunto finito e E é um conjunto de pares não-ordenados (resp. ordenados) de V. Os elementos de V são chamados de *vértices* e os de E são chamados *arestas* do grafo G.

Dado um grafo não-orientado $G = (V, E)$, dizemos que uma aresta $e = \{u, v\}$ tem *extremidades* u e v. Dado um vértice $v \in V$ (resp. conjunto $S \subseteq V$), denotamos por $\delta(v)$ (resp. $\delta(S)$) o conjunto de arestas com exatamente uma extremidade em v (resp. S).

Dado um grafo orientado $G = (V, E)$, dizemos que uma aresta $e = (u, v)$ *entra* em v e *sai* de u. Vamos denotar por $\delta^+(v)$ (resp. $\delta^-(v)$) o conjunto de arestas que saem de (resp. entram em) v. Dado um conjunto de vértices $S \subseteq (V)$, denotamos por $\delta^+(S)$

(resp. $\delta^-(S)$) o conjunto de arestas que saem de (resp. entram em) algum vértice de S e entram em (resp. saem de) algum vértice em $V \setminus S$.

Dado um grafo $G = (V, E)$, dizemos que $H = (V', E')$ é um *subgrafo* de G, se H é um grafo e $V' \subseteq V$ e $E' \subseteq E$.

Dado um grafo não-orientado (resp. orientado) $G = (V, E)$, um *passeio* é uma seqüência $P = (v_0, e_1, v_1, \ldots, e_k, v_k)$, onde $v_i \in V$, $e_i \in E$ e $e_i = \{v_{i-1}, v_i\}$ (resp. $e_i = (v_{i-1}, v_i)$) $i = 1, \ldots, k$. Neste caso, dizemos que P é um passeio de v_0 a v_k. Um *caminho* em G é um passeio onde todos os vértices são distintos. Um *circuito* é um passeio C onde todos os vértices são distintos, exceto o último que é igual ao primeiro. Um circuito é dito ser *hamiltoniano* se contém todos os vértices de G.

Dizemos que um grafo não-orientado G é *conexo* (resp. *completo*) se existe um caminho (resp. aresta) entre qualquer par de vértices de G. O grafo G é dito ser uma *árvore* se é conexo e sem circuitos.

Dado um grafo $G = (V, E)$, uma função de custo nas arestas $w : E \to \mathbb{Q}$ e um caminho P (resp. subgrafo, circuito, árvore), seu custo é denotado por $w(P)$ e é igual a soma dos custos das arestas em P.

O leitor interessado em saber mais sobre a teoria dos grafos, recomendamos o livro de Bondy e Murty [Bondy and Murty, 1976].

2.2. Complexidade Computacional

Muitos dos problemas que tratamos neste texto são problemas da classe $\mathcal{NP}$-difícil. Neste caso, não são conhecidos algoritmos *eficientes*, nem se espera que existam, para o atual modelo de computação. A seguir damos uma breve noção desta justificativa, que está baseada na teoria de $\mathcal{NP}$-completude e no modelo de máquinas de Turing. Para maiores detalhes, veja [Cormen et al., 2001, Garey and Johnson, 1979].

2.2.1. Problemas de Decisão

Durante muitos anos, pesquisadores tentaram independentemente desenvolver algoritmos rápidos para vários problemas de diversos tipos, sem sucesso. Esta dificuldade pode ser melhor compreendida após os trabalhos de Cook [Cook, 1971] e Karp [Karp, 1972], que montaram a base da teoria de $\mathcal{NP}$-completude, a qual resumiremos a seguir.

Para comparar os tempos gastos por um algoritmo, vamos definir o tamanho de uma instância I, para um problema, como o número de símbolos usados para representar a instância I através de um alfabeto com pelo menos dois símbolos. Para nossos propósitos, o tamanho de I pode ser considerado como o número de bits para representar I.

Com isso podemos definir um bem conhecido problema difícil, o *Problema do Circuito Hamiltoniano (CH)*.

> **Problema** CH(G): *Dado um grafo G, encontrar um circuito hamiltoniano em G, se existir.*

Vamos dizer que um algoritmo é *eficiente*, ou rápido, se este resolve o problema em tempo que é limitado por uma função polinomial no tamanho da instância. No caso

do problema CH, a instância é o grafo de entrada representado pelo conjunto de vértices e arestas.

Como muitos problemas tem como resposta soluções muitas vezes de difícil comparação, Cook limitou-se a problemas de decisão. I.e., problemas para os quais a resposta é simplesmente SIM ou NÃO. A versão de decisão do problema CH é apresentada a seguir.

> **Problema** DCH(G): *Dado um grafo G, decidir se existe um circuito hamiltoniano em G.*

Apesar do problema CH parecer ser mais difícil que o problema DCH, uma vez que um algoritmo para encontrar um circuito hamiltoniano nos leva diretamente a um algoritmo para decidir o segundo problema, a versão de decisão ainda conserva muito da dificuldade de resolução da versão não decisória. Para ilustrar este fato, vamos mostrar que com uma pequena perda no tempo computacional é possível transformar um algoritmo da versão de decisão para um algoritmo da versão não decisória. Para isto, considere um algoritmo $\mathcal{A}_{\text{DCH}}$ para o problema DCH e um grafo G.

Caso a chamada $\mathcal{A}_{\text{DCH}}(G)$ devolva NÃO, claramente não há circuito em G e portanto não há solução para o problema CH com a instância G.

Caso $\mathcal{A}_{\text{DCH}}(G)$ devolva SIM, existe um circuito em G. Neste caso é necessário encontrar tal circuito e devolvê-lo como resposta ao problema CH. Para isso, considere $E = \{e_1, \ldots, e_m\}$ e um grafo H_0 igual ao grafo G. Obtenha uma seqüência de grafos $H_1, \ldots, H_m$, onde o grafo H_i é obtido modificando o grafo H_{i-1} da seguinte maneira:

$$H_i \leftarrow \begin{cases} H_{i-1} - e_i & \text{se } \mathcal{A}_{\text{DCH}}(H_{i-1} - e_i) = \text{SIM}, \\ H_{i-1} & \text{caso contrário.} \end{cases}$$

Em cada iteração, testamos se a remoção de uma aresta mantém um circuito hamiltoniano no grafo resultante. Em caso afirmativo, podemos descartar a aresta, caso contrário a aresta deve fazer parte de um circuito. Claramente o grafo H_m é um circuito hamiltoniano. Assim, é possível obter um algoritmo para o problema CH executando o algoritmo $\mathcal{A}_{\text{DCH}}$ m vezes. Vamos denotar tal algoritmo como $\mathcal{A}_{\text{CH}}$. Desta forma, se o algoritmo $\mathcal{A}_{\text{DCH}}$ é de tempo polinomial, o algoritmo $\mathcal{A}_{\text{CH}}$ também é de tempo polinomial.

2.2.2. Redução entre problemas

A classe $\mathcal{NP}$ pode ser entendida como a classe dos problemas de decisão para os quais existe uma comprovação através de um certificado quando a resposta para o problema é SIM. Tal certificado deve ser de tamanho polinomial no tamanho da instância e deve poder ser verificado em tempo polinomial.

Por exemplo, considere um grafo de entrada G para o problema DCH. Para que o problema DCH esteja em $\mathcal{NP}$, deve existir um certificado de tamanho polinomial no tamanho de G, e um algoritmo de verificação do certificado que comprove que a resposta ao problema é de fato SIM. Tal certificado é uma prova que o grafo G de fato tem circuito hamiltoniano. Um exemplo de certificado para o DCH pode ser um circuito hamiltoniano C em G. Claramente a apresentação de C certifica que a resposta do problema é de fato

SIM, além disso, este certificado (o circuito) pode ser verificado em tempo polinomial. Nesta verificação, basta mostrar que C é de fato um circuito hamiltoniano válido de G. Note que não é necessário encontrar o certificado C, basta saber verificá-lo.

Note que $\mathcal{NP}$ representa uma classe grande. Por exemplo, estão nesta classe todos os problemas de decisão para os quais é possível testar a solução da correspondente versão não decisória em tempo polinomial.

A classe $\mathcal{P}$ contém os problemas em $\mathcal{NP}$ que podem ser decididos por um algoritmo de tempo polinomial.

A grande questão nesta teoria é saber se verificar uma solução é de fato muito mais fácil que encontrar uma solução. Assim, surgiu a conjetura "$\mathcal{P} = \mathcal{NP}$?". A maioria dos pesquisadores acredita que $\mathcal{P}$ e $\mathcal{NP}$ não sejam o mesmo conjunto. Uma das razões para se acreditar nisto é a existência de uma subclasse de $\mathcal{NP}$ chamada $\mathcal{NP}$-completo. Para definir esta classe, precisamos primeiro definir redução entre problemas. Vamos dizer que uma instância para um problema em $\mathcal{NP}$ tem solução se e somente se a resposta do problema à instância é SIM. Dizemos que um problema P é *redutível* em tempo polinomial a um problema Q, $P \preceq Q$, se existe um algoritmo de tempo polinomial que transforma qualquer instância I_P de P para alguma instância I_Q de Q tal que I_P tem solução se e somente se I_Q tem solução.

Duas importantes conseqüências que podemos tirar de uma redução $A \preceq B$ são as seguintes: *(a)* A existência de um algoritmo polinomial para decidir B implica também em um algoritmo polinomial para decidir A. *(b)* Se for provado que A não pode ser resolvido em tempo polinomial, então B também não pode ser resolvido em tempo polinomial.

Um problema $P \in \mathcal{NP}$ está em $\mathcal{NP}$-completo se todo problema de $\mathcal{NP}$ pode ser reduzido para P. De certa forma, a classe $\mathcal{NP}$-completo contém os problemas mais difíceis de $\mathcal{NP}$, já que a resolução de um problema $\mathcal{NP}$-completo em tempo polinomial implica que todos os problemas de $\mathcal{NP}$ podem ser resolvidos em tempo polinomial*. Cook [Cook, 1971] mostrou que um certo problema de satisfatibilidade de fórmulas booleanas, chamado SAT, é $\mathcal{NP}$-completo. Uma vez que temos um problema $\mathcal{NP}$-completo, a busca de outros problemas $\mathcal{NP}$-completos é facilitada. Para mostrar que um problema em $\mathcal{NP}$ é $\mathcal{NP}$-completo, basta pegar um problema já conhecido ser $\mathcal{NP}$-completo e reduzi-lo para o novo problema. Desta forma, Karp [Karp, 1972] mostrou que vários outros problemas também são $\mathcal{NP}$-completos, um destes problemas é o DCH. Desde então muitos problemas de interesse tem sido mostrados serem $\mathcal{NP}$-completos.

Dizemos que um problema P, não necessariamente de decisão, pertence à classe $\mathcal{NP}$-difícil se qualquer problema de $\mathcal{NP}$ pode ser reduzido a P em tempo polinomial. Como o problema DCH é $\mathcal{NP}$-completo, o problema CH é um problema $\mathcal{NP}$-difícil.

Uma possível configuração para estas classes de problemas é a seguinte:

*Lembre-se que a classe $\mathcal{NP}$ é grande.

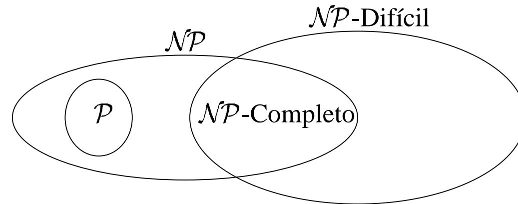


Figura 1: Possível configuração para as classes $\mathcal{P}$, $\mathcal{NP}$ e $\mathcal{NP}$-difícil.

2.2.3. Algoritmos Eficientes

Para muitos dos problemas de interesse, algoritmos polinomiais são sinônimos de algoritmos eficientes. Apesar de desejarmos tais algoritmos, acredita-se que os algoritmos para resolver problemas $\mathcal{NP}$-difíceis, devam ser pelo menos de tempo exponencial.

Para se ter uma idéia do quão rápido crescem as funções exponenciais, vamos supor que temos um supercomputador com velocidade de processamento de um Terahertz.† Além disso, vamos supor que temos um programa cuja quantidade de instruções executadas é dado por uma função $f(n)$, onde n é o tamanho da instância.

A tabela 1 ilustra o tempo necessário para algumas funções polinomiais e exponenciais neste supercomputador. Usamos as abreviações seg para segundos e séc para séculos.

$f(n)$	$n = 20$	$n = 40$	$n = 60$	$n = 80$	$n = 100$
n	$2{,}0\times10^{-11}$ seg	$4{,}0\times10^{-11}$ seg	$6{,}0\times10^{-11}$ seg	$8{,}0\times10^{-11}$ seg	$1{,}0\times10^{-10}$ seg
n^2	$4{,}0\times10^{-10}$ seg	$1{,}6\times10^{-9}$ seg	$3{,}6\times10^{-9}$ seg	$6{,}4\times10^{-9}$ seg	$1{,}0\times10^{-8}$ seg
n^3	$8{,}0\times10^{-9}$ seg	$6{,}4\times10^{-8}$ seg	$2{,}2\times10^{-7}$ seg	$5{,}1\times10^{-7}$ seg	$1{,}0\times10^{-6}$ seg
n^5	$2{,}2\times10^{-6}$ seg	$1{,}0\times10^{-4}$ seg	$7{,}8\times10^{-4}$ seg	$3{,}3\times10^{-3}$ seg	$1{,}0\times10^{-2}$ seg
2^n	$1{,}0\times10^{-6}$ seg	1,0 seg	13,3 dias	$1{,}3\times10^{5}$ séc	$1{,}4\times10^{11}$ séc
3^n	$3{,}4\times10^{-3}$ seg	140,7 dias	$1{,}3\times10^{7}$ séc	$1{,}7\times10^{19}$ séc	$5{,}9\times10^{28}$ séc

Tabela 1: Comparando tempos polinomiais e exponenciais.

Nota se que programas cujo tempo de processamento obedecem a funções exponenciais, como os da tabela, não são práticos, mesmo para instâncias pequenas.

Apesar da dificuldade em se resolver problemas $\mathcal{NP}$-difíceis, muitos destes problemas tem grande interesse prático e necessitam de uma solução. A tabela acima mostra que o investimento em um supercomputador pode não ser suficiente. Porém, há diversas técnicas que fazem deste cenário menos árduo, mas ainda assim bastante difícil.

Nas próximas seções veremos algumas técnicas e problemas resolvidos de maneira eficiente e outros para os quais não são esperados tais algoritmos.

†Mil vezes mais rápido que um computador de 1 Gigahertz.

3. Problemas Resolvidos com Programação Linear

Uma das principais ferramentas usadas no desenvolvimento de algoritmos para problemas de otimização é a programação linear. Um programa linear pode ser dado da seguinte forma:

> **Problema** $\text{PL}(A, b, c)$: *Dados matriz* $A = (a_{ij}) \in \mathbb{Q}^{m \times n}$*, vetores* $c = (c_i) \in \mathbb{Q}^n$ *e* $b = (b_i) \in \mathbb{Q}^m$*, encontrar vetor* $x = (x_i) \in \mathbb{Q}^n$ *(se existir) que*
>
> $$\text{minimize} \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{1}$$
>
> $$\text{sujeito a} \quad \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \leq & b_1\,, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq & b_2\,, \\ \vdots & \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & = & b_m\,. \end{cases} \tag{2}$$

A linha (1) representa a função objetivo do programa linear, que pode ser tanto de minimização como de maximização. Em (2) temos as restrições do problema, que podem ser tanto igualdades como desigualdades. Tanto a função objetivo como as restrições devem ser lineares e as variáveis x_i são definidas no conjunto dos racionais.

O conjunto de pontos que satisfazem todas as restrições é chamado de *poliedro* do programa linear. Dentre todos os pontos deste poliedro, as soluções do programa linear são aqueles pontos do poliedro que minimizam a função objetivo. Por uma tradição na área, as soluções de um programa linear são chamadas de *soluções ótimas* e os pontos que satisfazem todas as restrições, não necessariamente ótimas, de *soluções viáveis*.

EXEMPLO 3.1 *A figura 2 apresenta o poliedro do programa linear (3):*

$$\begin{array}{ll} \text{minimize} & -2x_1 - x_2 \\ \text{sujeito a} & \begin{cases} x_1 & +x_2 & \leq & 5\,, \\ 2x_1 & +3x_2 & \leq & 12\,, \\ x_1 & & \leq & 4\,, \\ x_1 & & \geq & 0\,, \\ & x_2 & \geq & 0\,. \end{cases} \end{array} \tag{3}$$

Existem vários métodos para se resolver um programa linear. O mais conhecido é o *método Simplex*, desenvolvido por Dantzig [Dantzig, 1951, Dantzig, 1963] no fim da década de 40. O método Simplex não é de tempo polinomial mas é bastante rápido, com tempo médio polinomial. O primeiro algoritmo de tempo polinomial é devido a Khachiyan [Khachiyan, 1979] ao desenvolver uma variante do método dos Elipsóides, de problemas de programação não-linear. Apesar de polinomial, tal método não se mostrou prático. Karmarkar [Karmarkar, 1984] desenvolveu o Método dos Pontos Interiores, que além de polinomial é bastante rápido.

Alguns dos problemas que veremos nesta seção usam algumas propriedades das soluções de um programa linear. A seguir, apresentamos alguns pontos desta teoria. O leitor interessado poderá encontrar mais detalhes sobre esta teoria nos seguintes livros [Chvátal, 1983, Ferreira and Wakabayashi, 1996, Papadimitriou and Steiglitz, 1982, Schrijver, 1986].

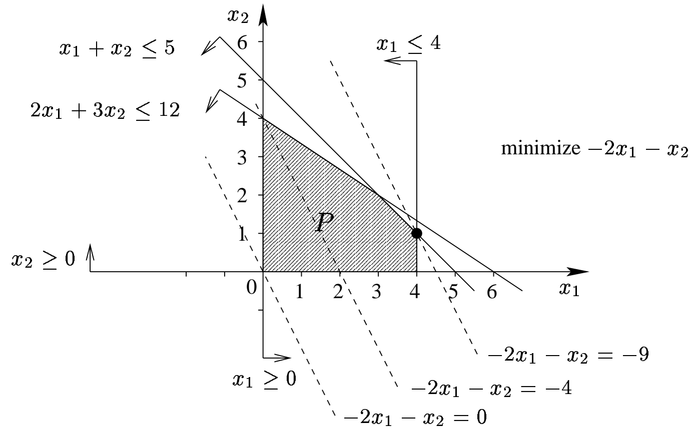


Figura 2: Programa linear com solução ótima $(x_1 = 4, x_2 = 1)$ de valor -9.

Dado pontos $y_i \in \mathbb{R}^d$ e $\alpha_i \in \mathbb{R}$, $i = 1, \ldots, n$ e $d \geq 1$, dizemos que $y = \alpha_1 y_1 + \alpha_2 y_2 + \cdots + \alpha_n y_n$ é uma *combinação convexa* dos pontos $\{y_1, \ldots, y_n\}$ se $\alpha_i \geq 0$ e $\alpha_1 + \cdots + \alpha_n = 1$.

EXEMPLO 3.2 *Os pontos que são combinação convexa de dois pontos x e y são:* $\text{conv}(\{x, y\}) := \{\alpha_1 x + \alpha_2 y : \ \alpha_1 \geq 0, \ \alpha_2 \geq 0, \ \alpha_1 + \alpha_2 = 1\}$. *Substituindo* $\alpha_2 = 1 - \alpha_1$, *temos* $\text{conv}(\{x, y\}) := \{\alpha x + (1-\alpha) y : \ 0 \leq \alpha \leq 1\}$. *Na figura 3, o ponto j é combinação convexa de x e y. Quando α tende a 0, a combinação convexa se aproxima de y e quando α tende a 1, a combinação convexa tende a x.*

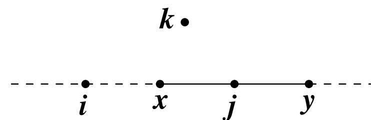


Figura 3: O ponto j é combinação convexa de x e y, mas i e k não são.

Os *vértices* ou *pontos extremais* de P são os pontos de P que não podem ser escritos como combinação convexa de outros pontos de P.

EXEMPLO 3.3 *Os pontos v_1, v_2, v_3, v_4, v_5 da figura 4 são vértices de P.*

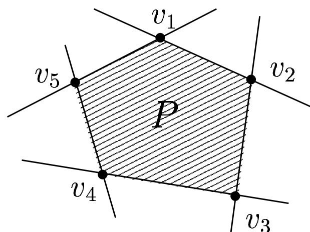


Figura 4: Vértices de um poliedro.

O seguinte teorema nos garante que soluções ótimas que são vértices (*vértices ótimos*) do poliedro de programação linear podem ser obtidas eficientemente.

TEOREMA 3.1 *Uma solução ótima de um programa linear que é vértice pode ser encontrada em tempo polinomial.*

Em geral os algoritmos para resolver programas lineares prevêem que o método devolva uma solução que é vértice. O método Simplex, por exemplo, encontra uma solução ótima percorrendo os vértices do poliedro.

Um poliedro $P \subseteq \mathbb{R}^n$ é dito ser *limitado*, se existe $M \in \mathbb{R}^+$ tal que $-M \leq x_i \leq M$ para todo $x \in P$.

Dado um conjunto de pontos S em $\mathbb{R}^n$, definimos o *fecho convexo*, denotado por *conv*(S), o conjunto dos pontos formados pela combinação convexa dos pontos de S. Dado um poliedro P, definimos seu *fecho inteiro*, P_I, como sendo o fecho convexo dos vetores inteiros de P. I.e., $P_I := conv(\{x \in P : x \text{ é inteiro}\})$.

EXEMPLO 3.4 *Na figura 5 temos (a) um exemplo de poliedro, (b) seus pontos inteiros e (c) o fecho inteiro do poliedro.*

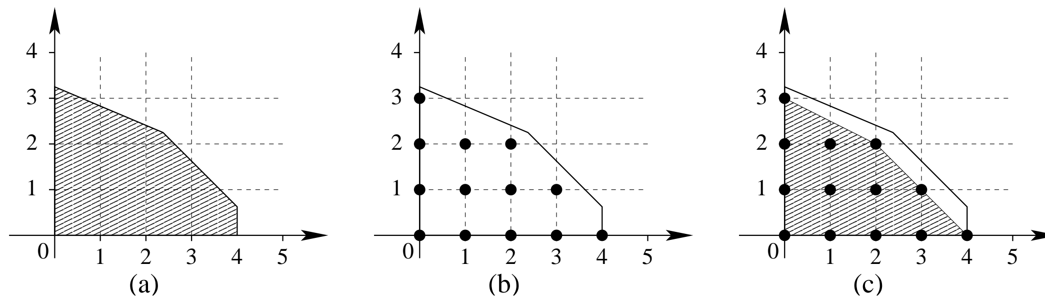
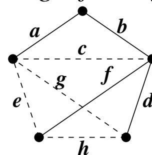


Figura 5: (a) Poliedro, (b) pontos inteiros e (c) fecho inteiro.

Dado um conjunto finito e ordenado E, dizemos que $\chi^A := (\chi^A_e : e \in E)$ é o vetor de incidência de A, $A \subseteq E$; onde $\chi^A_e = 1$ se $e \in A$ e 0 caso contrário.

EXEMPLO 3.5 *Seja* $G = (V, E)$ *o grafo abaixo com ordenação das arestas* $E = (a, b, c, d, e, f, g, h)$ *e* T *o subgrafo definido pelas arestas não tracejadas.*



Então o vetor de incidência das arestas de T *em* E *é o vetor* $\chi^T = (1, 1, 0, 1, 0, 1, 0)$.

3.1. Problema do Fluxo de Custo Mínimo

O problema do fluxo de custo mínimo é um problema que tem muitas aplicações e generaliza vários problemas. Um exemplo onde este problema pode ser aplicado, é na distribuição de produtos. Considere uma empresa que fabrica um certo produto em fábricas instaladas em diferentes cidades. Cada fábrica tem uma capacidade própria de produção que é totalmente vendida em centros consumidores (cidades). A distribuição dos produtos é feita através de uma rede ferroviária com linhas que ligam as cidades, cada linha possui uma capacidade de distribuição e um custo de transporte que é proporcional à quantidade de produtos transportados por ela. O problema consiste em encontrar uma distribuição da produção gastando o mínimo para transportar os produtos.

Seja $G = (V, E)$ um grafo orientado com função de capacidades nas arestas $c : E \rightarrow \mathbb{Q}^+$, demandas $d : V \rightarrow \mathbb{Q}$ e custos nas arestas $w : E \rightarrow \mathbb{Q}^+$. Dado um vértice

$v \in V$, se $d_v < 0$ dizemos que v é um *consumidor* e se $d_v > 0$ dizemos que v é um *produtor*. Dizemos que uma atribuição $x : E \rightarrow \mathbb{Q}^+$ é um *fluxo* em G se $\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = d_v$ para todo $v \in V$ e $0 \leq x_e \leq c_e$ para todo $e \in E$. É claro que para existir um fluxo, deve valer que $\sum_{v \in V} d_v = 0$.

EXEMPLO 3.6 *A figura 6-(a) apresenta um grafo com capacidades nas arestas e demandas nos vértices. A figura 6-(b) apresenta um fluxo satisfazendo as restrições de capacidade e demanda.*

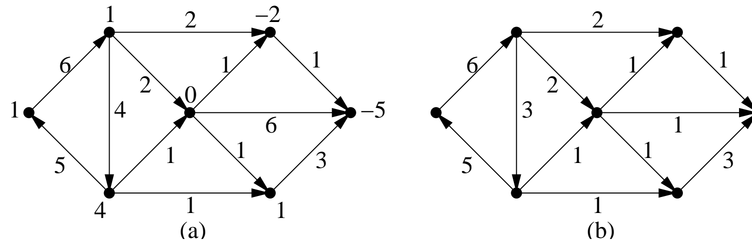


Figura 6: (a) Grafo com demandas e capacidades. (b) Fluxo viável.

Dado um fluxo $x : E \rightarrow \mathbb{Q}^+$ e uma função de custo nas arestas, $w : E \rightarrow \mathbb{Q}^+$, definimos o *custo do fluxo* x como $\sum_{e \in E} w_e x_e$.

> **Problema** DO FLUXO DE CUSTO MÍNIMO(G, c, d, w): *Dados um grafo orientado $G = (V, E)$, capacidades $c : E \rightarrow \mathbb{Q}^+$, demandas $d : V \rightarrow \mathbb{Q}^+$ e uma função de custo nas arestas $w : E \rightarrow \mathbb{Q}^+$, encontrar um fluxo $x : E \rightarrow \mathbb{Q}^+$ de custo mínimo.*

A formulação consiste em encontrar x que

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{e \in E} w_e x_e \\ \text{sujeito a} & \left\{ \begin{array}{l} \displaystyle\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = d_v \quad \forall v \in V\,, \\ \qquad\qquad 0 \leq x_e \leq c_e \quad \forall e \in E\,. \end{array} \right. \end{array} \tag{4}$$

O seguinte teorema nos dá uma informação importante acerca das soluções inteiras deste programa linear.

TEOREMA 3.2 *Se as capacidades nas arestas e as demandas dos vértices são inteiros, então os vértices do poliedro do fluxo são inteiros.*

3.2. Problema do Caminho de Peso Mínimo

O problema do caminho mínimo consiste no seguinte:

> **Problema** DO CAMINHO MÍNIMO: *Dado um grafo orientado $G = (V, E)$, custos nas arestas $w : E \rightarrow \mathbb{Q}^+$ e vértices s e t, encontrar um caminho de s para t de peso mínimo.*

EXEMPLO 3.7 *A figura 7-(a) apresenta um grafo com pesos nas arestas e a figura (b) apresenta um caminho de peso mínimo entre s e t.*

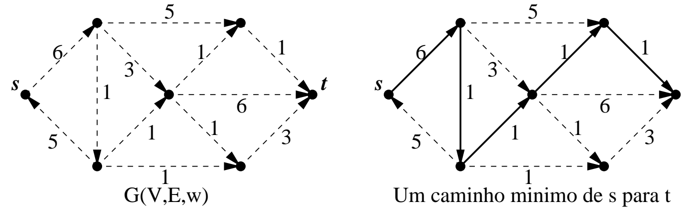


Figura 7: (a) Grafo com pesos nas arestas. (b) Caminho mínimo de s para t.

Este problema apresenta aplicações na determinação de rotas de custo mínimo, segmentação de imagens, reconhecimento de fala, etc.

O problema do caminho mínimo pode ser formulado como um caso especial do problema de fluxo de custo mínimo. Isto é, encontre x que

$$
\begin{array}{ll}
\text{minimize} & \sum_{e\in E} w_e x_e \\
\text{sujeito a} & \left\{ \begin{array}{rcll}
\sum_{e\in\delta^+(v)} x_e - \sum_{e\in\delta^-(v)} x_e & = & 0 & \forall v \in V \setminus \{s,t\}\,, \\
\sum_{e\in\delta^+(s)} x_e - \sum_{e\in\delta^-(s)} x_e & = & 1\,, & \\
\sum_{e\in\delta^+(t)} x_e - \sum_{e\in\delta^-(t)} x_e & = & -1\,, & \\
0 \quad \leq \quad x_e & \leq & 1 & \forall e \in E\,.
\end{array} \right.
\end{array} \tag{5}
$$

Aplicando o teorema 3.2 obtemos o seguinte resultado:

TEOREMA 3.3 *Se x é um vértice ótimo do poliedro em (5), então as arestas $e \in E$ onde $x_e = 1$ formam um caminho mínimo ligando s a t.*

3.3. Problema do Fluxo Máximo de s para t (st-fluxo máximo)

Este problema é um caso particular do Fluxo de Custo Mínimo. Neste caso não temos demandas nem custo nas arestas e temos apenas um vértice s *produzindo* fluxo e um vértice t *consumindo* o fluxo. O objetivo neste problema é maximizar o fluxo de s para t, respeitando as restrições de capacidade do fluxo.

EXEMPLO 3.8 *A figura 8-(a) apresenta um grafo de entrada, os vértices s e t e as capacidades nas arestas. A figura 8-(b) apresenta um fluxo de valor máximo.*

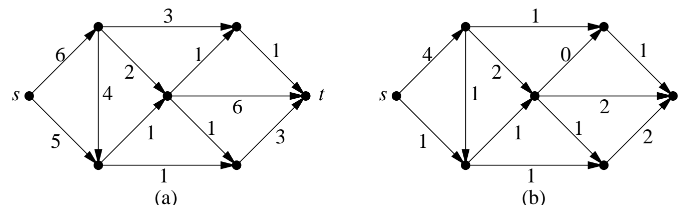


Figura 8: (a) Grafo com capacidades e demandas, (b) st-fluxo de valor máximo.

Seja $G = (V, E)$ um grafo orientado, capacidades $c : E \to \mathbb{Q}^+$ e vértices s e t. Dizemos que $x : E \to \mathbb{Q}^+$ é um *st-fluxo* se $\sum_{e\in\delta^+(v)} x_e - \sum_{e\in\delta^-(v)} x_e = 0 \quad \forall v \in V \setminus \{s,t\}$ e $0 \leq x_e \leq c_e \quad \forall e \in E$. Dado um st-fluxo x, definimos o *valor do fluxo x* como sendo $\sum_{e\in\delta^+(s)} x_e - \sum_{e\in\delta^-(s)} x_e$.

> **Problema** DO st-FLUXO MÁXIMO(G, c, s, t): *Dados um grafo orientado $G = (V, E)$, capacidades $c : E \to \mathbb{Q}^+$ e vértices s e t, encontrar um fluxo de s para t de valor máximo.*

Para apresentar uma formulação para este problema, vamos colocá-lo no formato do problema de fluxo de custo mínimo. Para isso, adicione uma aresta $t \to s$ em G, com custo -1 e capacidade suficientemente grande (*e.g.* soma das capacidades de todas as arestas), e coloque demandas e custos iguais a 0 para o restante do grafo. Assim, encontrar o fluxo de custo mínimo nesta instância é o mesmo que maximizar o fluxo de s para t no problema original.

EXEMPLO 3.9 *A figura 9 apresenta a transformação do grafo de entrada apresentado na figura 8 no formato do problema do fluxo de custo mínimo.*

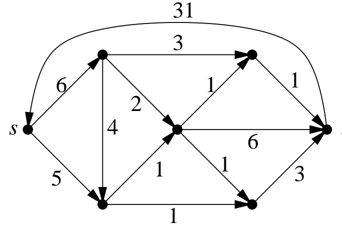


Figura 9: Transformação da instância da fig. 8-(a) para fluxo de custo mínimo.

A formulação do problema do st-fluxo máximo consiste em encontrar x que

$$\begin{array}{ll} \text{minimize} & -x_{ts} \\ \text{sujeito a} & \left\{ \begin{array}{rcll} \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e & = & 0 & \forall v \in V\,, \\ 0 \;\leq\; x_e & \leq & c_e & \forall e \in E\,. \end{array} \right. \end{array}$$

O próximo resultado segue direto da aplicação do teorema 3.2.

COROLÁRIO 3.4 *Se as capacidades c_e são inteiras, então os vértices do poliedro do st-fluxo são inteiros.*

3.4. Problema do Corte de Peso Mínimo

Seja $G = (V, E)$ um grafo orientado com capacidades $c : E \to \mathbb{Q}^+$ e vértices s e t. Um conjunto de arestas $\delta^+(S)$, $S \subseteq V$ é dito ser um *st-corte* se $s \in S$ e $t \notin S$. Por simplicidade, dado um conjunto de vértices S usamos tanto o termo corte para o conjunto de arestas $\delta^+(S)$ como para o conjunto de vértices S. Definimos a *capacidade de um st-corte* S, como sendo $c(\delta^+(S)) := \sum_{e \in \delta^+(S))} c_e$. O problema do st-corte mínimo pode ser definido da seguinte maneira:

> **Problema** DO st-CORTE MÍNIMO: *Dado um grafo orientado $G = (V, E)$, capacidades $c : E \to \mathbb{Q}^+$ e vértices s e t, encontrar um st-corte de capacidade mínima.*

Este problema encontra aplicações no projeto de redes de conectividade, classificação de dados *(data mining)*, particionamento de circuitos VLSI, etc.

Um dos resultados clássicos nesta área é o teorema do fluxo máximo/corte mínimo, apresentado a seguir.

TEOREMA 3.5 *O valor de um st-fluxo máximo de s para t é igual à capacidade de um st-corte mínimo.*

Um st-corte S de capacidade mínima pode ser obtido a partir de um fluxo x de valor máximo, começando S com o vértice s e acrescentando outros vértices que podem receber mais fluxo a partir de algum vértice de S. Um vértice $j \notin S$ pode receber mais fluxo de um vértice $i \in S$ se $(i, j) \in E$ e $x_{ij} < c_{ij}$ ou se $(j, i) \in E$ e $x_{ji} > 0$.

EXEMPLO 3.10 *A figura 10 apresenta (a) um grafo de entrada com capacidades nas arestas, (b) um fluxo de valor máximo do vértice s para o vértice t e (c) um corte mínimo separando s e t.*

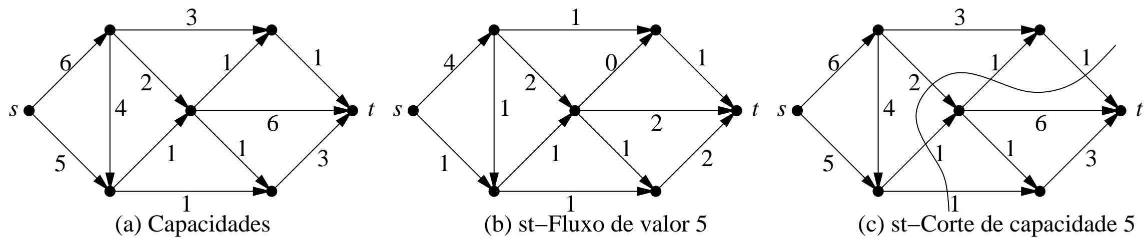


Figura 10: Fluxo máximo de s para t e corte mínimo separando s e t.

Os resultados que vimos envolvendo fluxos e corte foram para grafos orientados, mas os mesmos resultados podem ser obtidos para grafos não-orientados. De fato, os resultados para grafos não-orientados podem ser obtidos fazendo transformações para grafos orientados. Uma estratégia é trocar cada aresta sem orientação por duas arestas sobre os mesmos vértices, mas de orientações opostas.

4. Modelagem por Programação Linear Inteira

Na seção anterior, vimos o tratamento de vários problemas de resolução polinomial. Nesta seção também consideramos problemas $\mathcal{NP}$-difíceis. Para que possamos aplicar técnicas mais elaboradas, precisamos entender um pouco de modelagem dos problemas através de programação linear inteira.

Os problemas resolvidos anteriormente puderam ser resolvidos de maneira eficiente através do uso da programação linear. A pergunta que surge é se é possível usar programação linear também na resolução de problemas $\mathcal{NP}$-difíceis. A resposta, como seria de se esperar, é que sim. Programação linear também é uma das principais ferramentas no tratamento de problemas $\mathcal{NP}$-difíceis, tanto na obtenção de algoritmos exatos, heurísticos ou aproximados. Nesta seção apresentamos formulações para vários problemas. Um problema resolvido de forma eficiente, é o problema do emparelhamento de peso máximo. Todos os demais problemas apresentados nesta seção são $\mathcal{NP}$-difíceis.

Um programa linear inteiro é muito parecido com um programa linear. De fato a única diferença é a restrição adicional das variáveis serem inteiras.

Problema PLI(A, b, c): *Dados matriz* $A = (a_{ij}) \in \mathbb{Q}^{m \times n}$, *vetores* $c = (c_i) \in \mathbb{Q}^n$ *e* $b = (b_i) \in \mathbb{Q}^m$, *encontrar vetor* $x = (x_i) \in \mathbb{Z}^n$ *(se existir) que*

$$
\begin{array}{ll}
\text{minimize} & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\text{sujeito a} & \left\{ \begin{array}{rcl}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \leq & b_1 \,, \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq & b_2 \,, \\
\vdots & \vdots & \vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & = & b_m \,.
\end{array} \right.
\end{array}
$$

Aparentemente as diferenças com programas lineares são pequenas, mas em termos computacionais, há uma grande distância entre elas.

TEOREMA 4.1 *O problema de Programação linear inteiro é $\mathcal{NP}$-difícil.*

4.1. Modelagem através de variáveis binárias

O uso de variáveis binárias é uma das estratégias mais usadas para formular problemas de otimização. Neste caso, uma variável do programa linear pode assumir valor 0 ou 1. Em geral, a idéia principal consiste em definir variáveis $0/1$, digamos x_i, $i = 1, \ldots, n$, tal que se $x_i = 1$ então o objeto i pertence à solução, caso contrário o objeto i não pertence à solução.

4.1.1. Problema do Emparelhamento de Peso Máximo

Neste problema, temos que achar um conjunto de arestas de peso total máximo todas elas sem extremidades em comum. Dado um grafo $G = (V, E)$, dizemos que $M \subseteq E$ é um *emparelhamento* de G se M não tem arestas com extremos em comum.

Problema DO EMPARELHAMENTO DE PESO MÁXIMO(G, c): *Dados um grafo* $G = (V, E)$, *e custos nas arestas* $c : E \to \mathbb{Q}$, *encontrar emparelhamento* $M \subseteq E$ *que maximize* $c(M)$.

Algumas aplicações deste problema são as seguintes: atribuição de candidatos para vagas maximizando aptidão total, atribuição de motoristas de veículos, formação de equipes (*e.g.*, equipes de um chefe e vários subordinados), conexão em circuitos VLSI, etc.

Para formular este problema vamos usar variáveis binárias x_e para cada aresta $e \in E$. Nesta formulação, a variável x_e, para uma aresta $e = (u, v)$, tem valor 1 se (u, v) pertence ao emparelhamento e 0 caso contrário. Note que para definir um emparelhamento viável através das variáveis x_e, $e \in E$, a única restrição que temos que representar é que em nenhum vértice deve incidir mais que uma aresta. Assim, a formulação do problema de emparelhamento consiste em encontrar x que

$$
\begin{array}{ll}
\text{maximize} & \displaystyle\sum_{e \in E} c_e x_e \\
\text{sujeito a} & \left\{ \begin{array}{rcll}
\displaystyle\sum_{e \in \delta(v)} x_e & \leq & 1 & \forall v \in V \,, \\
x_e & \in & \{0, 1\} & \forall e \in E \,.
\end{array} \right.
\end{array}
\qquad (6)
$$

Como veremos posteriormente, este problema pode ser resolvido em tempo polinomial.

4.1.2. Problema da Mochila

Para entender este problema, considere a seguinte aplicação: Uma empresa pode investir até B reais em diferentes projetos $S = \{1, \ldots, n\}$, cada projeto i necessita de um investimento de s_i reais e tem um lucro esperado de v_i reais. O objetivo do problema é escolher os projetos que maximizam o lucro total sem ultrapassar o limite do orçamento. Mais formalmente, o problema da mochila pode ser definido como:

> **Problema** DA MOCHILA(B, n, s, v): *Dados itens* $S = \{1, \ldots, n\}$ *com valor* v_i *e tamanho* s_i *inteiros,* $i = 1, \ldots, n$*, e inteiro* B*, encontrar* $S' \subseteq S$ *que maximiza* $\sum_{i \in S'} v_i$ *tal que* $\sum_{i \in S'} s_i \leq B$.

Outras aplicações ocorrem em problemas de carregamento de veículos, problemas de corte e empacotamento, criptografia, etc.

Para formular o problema da mochila, vamos definir soluções através de variáveis binárias x_i, $i \in S$ tal que $x_i = 1$ indica que o elemento i pertence à solução e $x_i = 0$ indica que o elemento i não pertence à solução. O programa linear consiste em encontrar x que

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i \in S} v_i x_i \\ \text{sujeito a} & \left\{ \begin{array}{rcl} \displaystyle\sum_{i \in S} s_i x_i & \leq & B\,, \\ x_i & \in & \{0, 1\} \quad \forall i \in S\,, \end{array} \right. \end{array}$$

onde $S = \{1, \ldots, n\}$. Note que se uma variável x_i é igual a 1, seu peso s_i é contado no lado esquerdo da restrição e seu valor v_i é contado na função objetivo.

4.1.3. Problema da Localização de Recursos

Neste problema estamos interessados em localizar pontos de instalação de recursos para atender os elementos de um conjunto.

> **Problema** DA LOCALIZAÇÃO DE RECURSOS(n, m, f, c): *Dados potenciais recursos* $F = \{1, \ldots, n\}$*, clientes* $C = \{1, \ldots, m\}$*, custos* $f_i \geq 0$ *para instalar o recurso* i *e custos* $c_{ij} \geq 0$ *para um cliente* j *ser atendido pelo recurso* i*. Encontrar* $A \subseteq F$ *e atribuição* $\phi : C \to A$ *que minimiza o custo total para abrir os recursos em* A *e atender cada cliente* j *por* $\phi(j)$*,* $j \in C$.

Este problema encontra aplicações na instalação de postos de distribuição de mercadorias, centros de atendimento, instalação de antenas em telecomunicações, etc.

Note que aqui temos de determinar quais os recursos que iremos instalar e como conectar os clientes aos recursos instalados. Temos dois tipos de custos envolvidos. Se resolvermos instalar o recurso i, devemos pagar f_i pela sua instalação. Além disso, todos os clientes devem ser atendidos por algum recurso. Assim, pagamos um preço pela conexão estabelecida entre um cliente e o recurso instalado que esteja mais próximo. Vamos definir soluções através de variáveis binárias y_i, $i \in F$, e variáveis x_{ij}, $i \in F$ e $j \in C$. A igualdade $y_i = 1$ é equivalente a dizer que o recurso i foi escolhido para ser

instalado e a igualdade $x_{ij} = 1$ é equivalente a dizer que o cliente j será atendido pelo recurso i. Assim, o problema consiste em encontrar (x, y) que

$$
\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{i \in F} f_i y_i + \sum_{ij \in E} c_{ij} x_{ij} \\
\text{sujeito a} & \left\{ \begin{array}{rcll}
\displaystyle\sum_{ij \in E} x_{ij} & = & 1 & \forall j \in C\,, \\
x_{ij} & \leq & y_i & \forall ij \in E\,, \\
y_i & \in & \{0,1\} & \forall i \in F\,, \\
x_{ij} & \in & \{0,1\} & \forall i \in F \text{ e } j \in C\,.
\end{array} \right.
\end{array}
$$

A primeira restrição indica que todo cliente deve ser conectado a algum recurso. A segunda restrição indica que um cliente só deve ser conectado a um recurso instalado (se $y_i = 0$ esta restrição força que $x_{ij} = 0$ para todo $j \in C$).

4.1.4. Problema da Árvore de Steiner

Seja $G = (V, E)$ um grafo não-orientado e T um subconjunto de V.

> **Problema** DA ÁRVORE DE STEINER: *Dados um grafo $G = (V, E)$, um conjunto $T \subseteq V$ e uma função de custo nas arestas $c : E \to \mathbb{Q}^+$ encontrar uma árvore em G contendo todos os vértices de T e que seja de custo mínimo.*

Os vértices em T são chamados de vértices *terminais*. Neste caso, estamos procurando um subgrafo de G que conecta todos os terminais e é de custo mínimo.

Este problema encontra aplicações no roteamento de circuitos VLSI *(VLSI Layout and routing)*, projeto de redes de conectividade, determinação de amplificadores de sinal em redes óticas, construção de árvores filogenéticas, etc.

EXEMPLO 4.1 *No grafo da figura 11, os vértices quadrados representam os vértices terminais e o subgrafo definido pelas arestas sólidas uma solução para o problema da árvore de Steiner.*

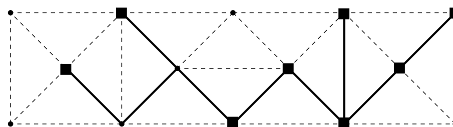


Figura 11: Grafo com árvore de Steiner em destaque.

Em muitos problemas que envolvem alguma estrutura de conectividade, é comum usarmos variáveis binárias para as arestas de conexão. Em geral as arestas tem custos que definem o custo de uma solução, o qual estamos querendo minimizar. Além disso, permitem facilmente escrever restrições relativas às restrições de conectividade que devem ser respeitadas.

Vamos definir as soluções através de variáveis binárias x_{ij} tal que $x_{ij} = 1$ se a aresta ij pertence à solução, caso contrário a aresta não pertence à solução.

Para formular este problema, usaremos o conceito de corte. Seja F um conjunto de arestas que não define uma solução viável para o problema de Steiner. Neste caso, devem existir vértices terminais s e t que não estão conectados através das arestas em F. Neste caso, é possível particionar o conjunto de vértices em duas partes S e $\overline{S}$, onde $\overline{S} = V \setminus S,\ s \in S,\ t \in \overline{S}$ e nenhuma aresta de F liga vértices de S a $\overline{S}$. Certamente, pelo menos uma das arestas do corte $\delta(S)$ deveria estar na solução. A formulação que veremos usa exatamente esta condição e inclui restrições que forçam que pelo menos uma aresta seja escolhida —para todo corte— separando dois terminais.

EXEMPLO 4.2 *A figura 12 mostra um grafo onde os vértices terminais são quadrados. A atribuição para x definida com $x_e = 1$ para as arestas sólidas e $x_e = 0$ para as arestas tracejadas não é uma solução viável, pois existe um corte que separa os dois terminais que está sendo violado. Havendo a desigualdade de corte $x_d + x_e + x_i + x_g + x_j \geq 1$ na formulação, tal atribuição inviável não seria permitida.*

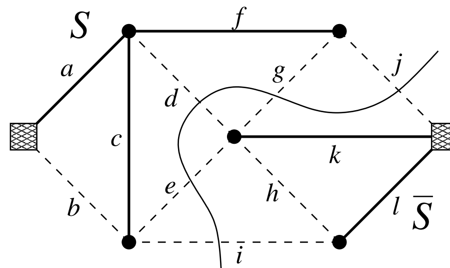


Figura 12: Atribuição inválida para o problema de Steiner.

Assim, esta formulação inclui todas as restrições de corte que separam pelo menos dois terminais. O problema consiste em encontrar x que

$$\begin{array}{ll} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{sujeito a} & \begin{cases} \sum_{e \in \delta(S)} x_e \geq 1 & \forall S \subset V: \quad S \cap T \neq T \quad \text{e} \quad S \cap T \neq \emptyset, \\ x_e \in \{0,1\} & \forall e \in E, \end{cases} \end{array} \tag{7}$$

onde T é o conjunto de vértices terminais.

A primeira restrição impõe que todo corte que separa dois terminais deve ter pelo menos uma aresta que pertença à solução. Apesar da quantidade de desigualdades envolvidas nesta formulação ser exponencial, veremos posteriormente que é possível aproveitar esta formulação de maneira mais prática.

4.1.5. Problema do Caixeiro Viajante

Este problema é conhecido na literatura como *Traveling Salesman Problem (TSP)* e é talvez o problema mais estudado na área de otimização combinatória. Esta atração se deve ao desafio que este problema tem sido para os pesquisadores durante vários séculos aliado a uma definição simples com várias aplicações práticas.

Problema $\text{TSP}(G, c)$: *Dados um grafo completo $G = (V, E)$ e um custo c_e em $\mathbb{Q}^+$ para cada aresta e, determinar um circuito hamiltoniano C que minimize $c(C)$.*

EXEMPLO 4.3 *Na figura 13, apresentamos a solução ótima de um circuito hamiltoniano para uma instância euclidiana de 52 pontos na cidade de Berlin.*

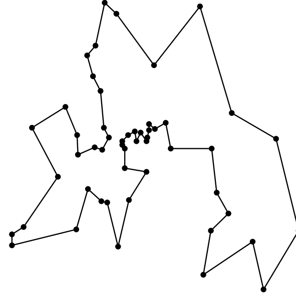


Figura 13: Circuito ótimo ligando 52 pontos na cidade de Berlin.

Algumas aplicações deste problema consistem em: perfuração e solda de circuitos impressos, determinação de rotas de custo mínimo, seqüenciamento de DNA, etc.

Para descrever a formulação, vamos definir as soluções através de variáveis binárias x_{ij}, tal que $x_{ij} = 1$ indica que a aresta ij pertence à solução e $x_{ij} = 0$ indica que a aresta ij não pertence à solução. A formulação que iremos apresentar está baseada em dois pontos: *(i)* a solução deve ser formada por circuitos disjuntos cobrindo todos os vértices e *(ii)* o solução deve ser conexa. Estes dois pontos forçam que a solução seja apenas um circuito.

No primeiro ponto, as restrições devem garantir que em todo vértice incidem exatamente duas arestas. Colocando apenas esta restrição, podemos obter uma atribuição que é um conjunto de circuitos disjuntos. É neste ponto que usamos restrições que forçam que a solução seja conexa. A estratégia é a mesma usada para o problema de Steiner, para garantir uma solução conexa. Neste caso, todo corte deve ter pelo menos duas arestas na solução. Podemos pensar que cada conjunto S, tal que $\emptyset \neq S \subset V$, divide o conjunto de vértices em duas partes, S e $V \setminus S$. Se estamos percorrendo o circuito em S, em algum momento o circuito deve ir para $V \setminus S$ e posteriormente deve voltar a S. Isto poderia ser repetido mais vezes, mas certamente deve ter pelo menos duas arestas da solução ligando S e $V \setminus S$. O programa linear consiste em encontrar x que

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{e \in E} c_e x_e \\ \text{sujeito a} & \left\{ \begin{array}{rcll} \displaystyle\sum_{e \in \delta(v)} x_e & = & 2 & \forall v \in V\,, \\ \displaystyle\sum_{e \in \delta(S)} x_e & \geq & 2 & \forall S \subset V, \quad S \neq \emptyset\,, \\ & & & \text{(restrições de corte)} \\ x_e & \in & \{0,1\} & \forall e \in E\,. \end{array} \right. \end{array} \tag{8}$$

4.2. Formulações com variáveis inteiras

Nesta seção vamos considerar problemas usando formulações por programas lineares inteiros com variáveis não necessariamente binárias.

4.2.1. Problema de Empacotamento

Uma empresa deve vender objetos $O = \{1, 2, \ldots, m\}$, cada objeto i com demanda d_i que devem ser recortados a partir de placas que devem ter uma configuração previamente estabelecida. Há um total de n configurações possíveis e cada configuração j tem a_{ij} items do objeto i. O objetivo é encontrar as quantidades que a empresa deve cortar de cada configuração para suprir a demanda, cortando o menor número possível de placas.

EXEMPLO 4.4 *Na figura 14 apresentamos quatro configurações possíveis de placas. Cada placa tem uma quantidade fixa de itens de cada objeto. A placa P3 tem 0 itens do objeto 1, 1 item do objeto 2, 3 itens do objeto 3 e 2 itens do objeto 4.*

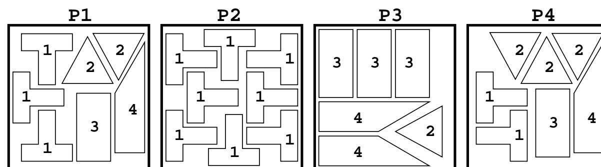


Figura 14: Configurações de placas a serem cortadas

Para formular este problema, vamos definir variáveis inteiras $x_j \geq 0$ para cada placa j. A variável x_j dá a quantidade de placas na configuração j que devemos cortar. Além disso, as placas a serem cortadas devem suprir a demanda do objeto i. Para satisfazer as demandas, devemos impor para cada objeto i, que a soma dos itens i cortados em todas as placas seja pelo menos sua demanda d_i. Isto é $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq d_i$.

EXEMPLO 4.5 *Considere as configurações apresentadas na figura anterior. Suponha que $d_1 = 950$, $d_2 = 1150$, $d_3 = 495$ e $d_4 = 450$. O programa linear consiste em encontrar x que*

$$\begin{array}{ll} \text{minimize} & x_1 + x_2 + x_3 + x_4 \\ \text{sujeito a} & \left\{ \begin{array}{lllllll} 3x_1 & +8x_2 & & +2x_4 & \geq & 950\,, & \\ 2x_1 & & +1x_3 & +3x_4 & \geq & 1150\,, & \\ 1x_1 & & +3x_3 & +1x_4 & \geq & 495\,, & \\ 1x_1 & & +2x_3 & +1x_4 & \geq & 450\,, & \\ & & & x_i & \in & \mathbb{Z}^*, & i = 1, \ldots, 4\,. \end{array} \right. \end{array} \tag{9}$$

Na forma geral, o programa linear consiste em encontrar x que

$$\begin{array}{ll} \text{minimize} & x_1 + x_2 + \cdots + x_n \\ \text{sujeito a} & \left\{ \begin{array}{lll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq & d_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq & d_2 \\ \vdots & \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq & d_m \\ x_i & \in & \mathbb{Z}^*, \quad i = 1, \ldots, n\,. \end{array} \right. \end{array}$$

Outras aplicações onde este problema ocorre é no corte de barras, alocação de comerciais de TV, alocação em páginas de memória, corte de placas (vidro, madeira, chapas, tecido, espuma, etc), empacotamento em contêineres, etc.

4.2.2. Truques de modelagem

Durante a formulação de um problema, podemos precisar de alguns *truques* para tratar certas condições. Veremos algumas destas e os truques que podemos aplicar.

Desigualdades excludentes: Suponha que em uma formulação deve ocorrer apenas uma entre duas desigualdades: ou vale $a'x \leq \alpha'$ ou vale $a''x \leq \alpha''$ mas ambas não podem ser válidas na formulação. Podemos implementar estas restrições usando uma nova variável binária, digamos Δ, com valor 0 se vale a primeira desigualdade e 1 caso a segunda deva ocorrer. As duas desigualdades, junto com estas condições pode ser formulada como segue.

$$\begin{array}{rrcl} a'x & -M\cdot\Delta & \leq & \alpha', \\ a''x & -M\cdot(1-\Delta) & \leq & \alpha'', \\ & \Delta & \in & \{0,1\}, \end{array}$$

onde M é um valor suficientemente grande.

Alternativas no lado direito de igualdades: Se deve valer a igualdade $ax = \alpha$ onde $\alpha \in \{\alpha_1, \ldots, \alpha_k\}$, podemos usar novas variáveis binárias $\Delta_1, \ldots, \Delta_k$ onde $\Delta_i = 1$ se e somente se a igualdade satisfeita é $ax = \alpha_i$. Assim, este tipo de restrição pode ser implementada como segue:

$$\begin{array}{lcl} ax & = & \sum_{i=1}^{k} \alpha_i \Delta_i, \\ \sum_{i=1}^{k} \Delta_i & = & 1, \\ \Delta_i & \in & \{0,1\}. \end{array}$$

Penalidades em desigualdades: As vezes permitimos que uma desigualdade $ax \leq \alpha$ seja violada (ax fica maior que α), mas penalizamos a quantidade violada por um fator P. Este tipo de condição pode ser formulada inserindo uma variável que represente a parte violada e colocando um custo nesta variável dentro da função objetivo. Isto é,

$$\begin{array}{lccl} \text{minimize} & \ldots & + Py, \\ & ax & \leq & \alpha + y, \\ & y & \geq & 0. \end{array}$$

4.2.3. Atribuição de Freqüências

Vamos considerar um problema onde precisamos minimizar o número de freqüências usadas nas antenas, respeitando condições de distância entre pares de antenas.

> **Problema** DE ATRIBUIÇÃO DE FREQÜÊNCIAS(n, d): *Dados conjunto de antenas* $A = \{1, 2, \ldots, n\}$, *conjunto de freqüências* $F = \{1, 2, \ldots\}$, *e uma função distância* $d : A \times A \to \mathbb{N}$, *encontrar uma atribuição de freqüências para as antenas* $f : A \to F$ *tal que a distância das freqüências atribuídas para as antenas* i *e* j *é pelo menos* $d(i, j)$. *O objetivo é encontrar uma atribuição de freqüências viável que minimize o valor da maior freqüência usada.*

Para formular este problema, usaremos variáveis f_i indicando a freqüência atribuída à antena i. Cada freqüência deve pertencer ao intervalo $\{1, \ldots, K\}$: $1 \leq$

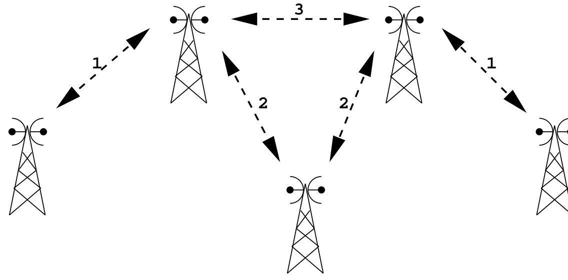


Figura 15: Antenas com freqüências que devem obedecer distâncias mínimas.

$f_i \leq K, \quad \forall i \in A$. Além disso, a atribuição deve satisfazer a função de distância: $|f_i - f_j| \geq d(i,j), \quad i \in A \quad e \quad j \in A$. Note que estas restrições não são lineares, mas podemos transformá-la em outras restrições lineares equivalentes da seguinte forma: Usamos uma variável binária Δ_{ij} para indicar as alternativas do módulo. Se $\Delta_{ij} = 0$ então deve ocorrer $f_i - f_j \geq d(i,j)$ e se $\Delta_{ij} = 1$ então deve ocorrer $f_j - f_i \geq d(i,j)$. Assim, podemos trocar a restrição não-linear

$$|f_i - f_j| \geq d(i,j)$$

pelas seguintes restrições lineares

$$\begin{array}{rrcl} f_i - f_j & +M\cdot\Delta_{ij} & \geq & d(i,j)\,, \\ f_j - f_i & +M\cdot(1-\Delta_{ij}) & \geq & d(i,j)\,, \\ & \Delta_{ij} & \in & \{0,1\}\,, \end{array}$$

onde M é um número suficientemente grande. Note que se $\Delta_{ij} = 0$, a primeira restrição nos dá $f_i - f_j \geq d(i,j)$ e conseqüentemente $f_i \geq f_j$; e a segunda restrição é sempre válida. O caso onde $\Delta_{ij} = 1$ tem análise análoga. Assim, o programa linear consiste em encontrar x que

$$\begin{array}{ll} \text{minimize} & K \\ \text{sujeito a} & \left\{ \begin{array}{rcll} f_i - f_j + M\cdot\Delta_{ij} & \geq & d(i,j) & \forall i \neq j, i \in A, j \in A\,, \\ f_j - f_i + M\cdot(1-\Delta_{ij}) & \geq & d(i,j) & \forall i \neq j, i \in A, j \in A\,, \\ \Delta_{ij} & \in & \{0,1\} & \forall i \neq j, i \in A, j \in A\,, \\ f_i & \geq & 1 & \forall i \in A\,, \\ f_i & \leq & K & \forall i \in A\,, \\ f_i & \in & \mathbb{Z} & \forall i \in A\,, \\ K & \in & \mathbb{Z}\,. & \end{array} \right. \end{array}$$

5. Técnicas para Tratamento de Programas Lineares Inteiros

Nesta seção consideramos várias técnicas para obter soluções a partir dos modelos de programação linear inteira.

5.1. Relaxando o Programa Linear Inteiro

Para alguns problemas, a simples estratégia de relaxar as condições de integralidade pode nos levar a boas informações, algumas vezes até a soluções ótimas. Este é o caso do problema de emparelhamento de peso máximo em grafos bipartidos.

5.1.1. Emparelhamento de Peso Máximo em Grafos Bipartidos

Veremos que é possível resolver o problema do emparelhamento de peso máximo de maneira simples e eficiente quando o grafo é bipartido. Um grafo $G = (V, E)$ é dito ser *bipartido* se V pode ser particionado em duas partes X e Y tal que todas as arestas de E tem extremidades em partes distintas.

EXEMPLO 5.1 *Na figura 16, apresentamos um grafo bipartido com um emparelhamento em destaque.*

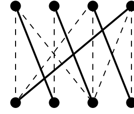


Figura 16: Arestas não tracejadas formam um emparelhamento do grafo.

Considere a formulação (6). Esta formulação nos dá pontos no espaço $\{0,1\}^E$, onde cada ponto é um emparelhamento possível. O fecho inteiro destes pontos, denotado pelo nome de *poliedro dos emparelhamentos* de um grafo $G = (V, E)$ é definido como

$$P_{Emp}(G) := conv(\{\chi^M \in \mathbb{R}^E : M \text{ é um emparelhamento em } G\}),$$

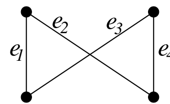
onde χ^M é o vetor de incidência do emparelhamento M.

Uma primeira tentativa para tratar este problema é aproximá-lo através da programação linear. Para isso, vamos relaxar a condição de integralidade de cada variável, pela pertinência no intervalo que contém os valores inteiros associados à variável. A este processo damos o nome de *relaxação linear* de uma formulação inteira. A relaxação linear da formulação (6), consiste em encontrar x que

$$(LP_{Emp}) \qquad \begin{array}{ll} \text{maximize} & \sum_{e \in E} c_e x_e \\ \text{sujeito a} & \begin{cases} \sum_{e \in \delta(v)} x_e \leq 1 & \forall v \in V, \\ 0 \leq x_e \leq 1 & \forall e \in E. \end{cases} \end{array} \qquad (10)$$

A primeira impressão é que o poliedro (LP_{Emp}) pode nos levar a soluções ótimas que não são inteiras. De fato, algumas soluções ótimas podem não ser inteiras, como mostra o seguinte exemplo.

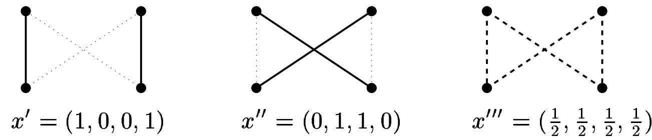
EXEMPLO 5.2 *Considere o seguinte grafo bipartido com custos unitários:*



A formulação (10) aplicado a este grafo, consiste em encontrar $x = (x_{e_1}, x_{e_2}, x_{e_3}, x_{e_4})$ que

$$\begin{array}{ll} \text{maximize} & x_{e_1} + x_{e_2} + x_{e_3} + x_{e_4} \\ \text{sujeito a} & \begin{cases} x_{e_1} + x_{e_2} \leq 1, & x_{e_3} + x_{e_4} \leq 1, \\ x_{e_1} + x_{e_3} \leq 1, & x_{e_2} + x_{e_4} \leq 1, \\ 0 \leq x_{e_1} \leq 1, & 0 \leq x_{e_2} \leq 1, \\ 0 \leq x_{e_3} \leq 1, & 0 \leq x_{e_4} \leq 1. \end{cases} \end{array} \qquad (11)$$

Os seguintes vetores são soluções ótimas deste programa linear:



Apesar da solução x''' ser solução ótima do programa (11), esta não é um vértice do poliedro, já as soluções x' e x'' o são. De fato, x''' é combinação convexa de x' e x'' ($x''' = \frac{1}{2}x' + \frac{1}{2}x''$).

Como podemos suspeitar, todos os vértices do poliedro da formulação (LP_{Emp}) —quando definida para grafos bipartidos— são inteiros. Assim, o seguinte resultado atribuído a Birkhoff [Birkhoff, 1946] e von Neumann [von Neumann, 1953] é válido.

TEOREMA 5.1 *Se G é um grafo bipartido, então $LP_{Emp}(G) = P_{Emp}(G)$.*

Com este teorema, basta encontrar uma solução ótima, que é vértice, do programa linear $(LP_{Emp}(G))$ para obter um emparelhamento em G. Apesar deste método ter dado resultados bons para o problema de emparelhamento em grafos bipartidos, não é para todos os problemas para os quais temos esta sorte. De fato, para todos os problemas $\mathcal{NP}$-difíceis, não é esperado que isto ocorra.

5.1.2. Obtendo delimitantes para a solução ótima

Mesmo que um poliedro não seja inteiro, ele pode nos dar informações úteis sobre seu correspondente fecho inteiro. Seja P um poliedro e I o conjunto de pontos inteiros em P. Denote por P_I o correspondente poliedro inteiro em P (i.e., $P_I := conv(I)$). Quando tratamos com programação linear inteira, queremos otimizar em P_I, mas em geral só temos P. Algumas considerações que podemos fazer acerca destes dois poliedros são as seguintes: *(i)* As soluções ótimas em P_I e P podem estar muito próximas. *(ii)* Todos os pontos de P_I pertencem a P.

Vamos denotar por $(P, c, \min)$ (resp. $(P_I, c, \min)$) o programa linear (resp. linear inteiro) de minimização sobre o poliedro P (resp. P_I) com a função objetivo definida pelo vetor de coeficientes c. Os seguintes resultados para um problema de minimização são válidos em relação a P e P_I.

PROPOSIÇÃO 5.2 *A solução ótima de $(P, c, \min)$ é menor ou igual que a solução ótima de $(P_I, c, \min)$.*

Este resultado nos ajuda a dizer mais sobre a qualidade de uma solução viável, uma vez que o valor da solução ótima está entre o valor da solução viável e o valor do programa relaxado.

PROPOSIÇÃO 5.3 *Se L é o valor ótimo de $(P, c, \min)$ (limitante inferior) e U é o valor de uma solução viável (limitante superior), não necessariamente ótima, para $(P_I, c, \min)$, então esta solução viável está a no máximo um fator U/L do valor da solução ótima.*

Resultados análogos podem ser obtidos para problemas de maximização.

5.2. Resolvendo Programação Linear Inteira por Arredondamento

Neste método, partimos de um modelo em programação linear inteira. Relaxamos a formulação e obtemos uma solução ótima para o problema relaxado. Em seguida, us-

amos uma estratégia para arredondar as variáveis fracionárias para que assumam valores inteiros. A seguir, veremos algumas destas estratégias.

5.2.1. Arredondamento Simples

Este método consiste simplesmente em arredondar as variáveis fracionárias para valores inteiros, de tal maneira a obter uma solução viável. No exemplo a seguir, aplicamos esta técnica ao problema de empacotamento, visto na seção 4.2.1.

EXEMPLO 5.3 *Considere a formulação (9) do exemplo 4.5. A relaxação linear desta formulação é a seguinte:*

$$\begin{array}{ll} \text{minimize} & x_1 + x_2 + x_3 + x_4 \\ \text{sujeito a} & \left\{ \begin{array}{lllllll} 3x_1 & +8x_2 & & +2x_4 & \geq & 950\,, \\ 2x_1 & & +1x_3 & +3x_4 & \geq & 1150\,, \\ 1x_1 & & +3x_3 & +1x_4 & \geq & 495\,, \\ 1x_1 & & +2x_3 & +1x_4 & \geq & 450\,, \\ x_i \geq 0\,. \end{array} \right. \end{array}$$

Resolvendo este programa linear, obtemos uma solução de valor $437{,}656$ *e valores* $x_1 = 0$*,* $x_2 = 26{,}406$*,* $x_3 = 41{,}875$ *e* $x_4 = 369{,}375$*. Como o número de placas em uma solução deve ser inteiro, uma solução ótima deve usar pelo menos 438 placas.*

Arredondando as variáveis fracionárias para cima, obtemos uma solução viável $\overline{x}$ *de valor* $\overline{x}_1 = 0$*,* $\overline{x}_2 = 27$*,* $\overline{x}_3 = 42$ *e* $\overline{x}_4 = 370$*, com um total de* 439 *placas. Assim, se esta solução não for ótima, ela usa no máximo uma placa a mais que uma solução ótima.*

5.2.2. Arredondamento Iterativo

Esta estratégia consiste em executar várias iterações de arredondamentos simples até que se obtenha uma solução viável, ou um sistema inviável. Cada iteração consiste em resolver o programa linear relaxado fixando permanentemente algumas variáveis para valores inteiros. A primeira iteração começa com a solução ótima fracionária da relaxação da formulação inteira.

Esta estratégia foi usada por Jain [Jain, 2001] no problema de projeto de redes com tolerância a falhas, que é uma generalização do problema da árvore de Steiner, apresentado na seção 4.1.4. Jain mostrou que em cada iteração, o correspondente programa relaxado ou é uma solução viável, ou tem pelo menos uma variável de valor fracionário pelo menos $1/2$. Assim, o algoritmo escolhe em cada iteração, todas as variáveis com valor pelo menos $1/2$ para serem arredondadas para 1. Neste artigo, Jain mostra que a solução obtida por este algoritmo é no máximo duas vezes pior que a solução ótima. Este é o melhor fator de aproximação teórico conhecido para este problema. O leitor poderá encontrar mais sobre algoritmos de aproximação em [Carvalho et al., 2001, Vazirani, 2000].

5.2.3. Arredondamento Probabilístico

Nesta estratégia, obtemos uma solução fracionária para o programa relaxado da formulação inteira. Em seguida, arredondamos as variáveis de maneira probabilística.

Este método foi introduzido por Raghavan e Thompson [Raghavan and Thompson, 1987] e vem sendo usado com sucesso para vários problemas. Um exemplo foi o usado por Goemans e Williamson [Goemans and Williamson, 1994] para o problema de satisfatibilidade máxima. Os autores formularam este problema através de um programa linear com variáveis binárias. Resolveram o programa relaxado e interpretaram as variáveis como probabilidades. Assim, uma variável do programa linear relaxado com valor $x_v = 0{,}8$, é arredondado para 1 com probabilidade 0,8. O algoritmo apresentado por eles além de ter boa aproximação, faz parte dos melhores algoritmos existentes para este problema.

5.3. Branch & Bound e Programação Linear Inteira

A estratégia de *Branch & Bound* consiste em enumerar o espaço de soluções através de uma árvore de enumeração e percorrer os ramos da árvore de forma sistemática, evitando percorrer ramos que levam a soluções inviáveis ou que não levam a soluções melhores que as já encontradas. Este é um método geral que pode ser aplicado a problemas de várias formas, não necessariamente de otimização.

Vamos ver uma simplificação do método Branch & Bound para obter soluções ótimas inteiras de um programa linear de minimização. A árvore de enumeração do método Branch & Bound tem um nó raiz que representa todo o espaço de soluções do problema. Além disso, cada nó da árvore representa um programa linear sendo que o nó raiz representa o programa linear relaxado.

O algoritmo mantém um conjunto de nós ativos, inicialmente apenas com o nó raiz, e a melhor solução viável y^* encontrada durante a busca. Em cada iteração do método, escolhe-se um nó Q para ser removido do conjunto de ativos e avaliado. Vamos denotar por $\mathrm{val}(y)$ o valor da solução y, caso y seja solução viável, ou ∞ se $y = \emptyset$; e OPT-LP$(Q, c, \min)$ uma solução ótima do programa linear $(Q, c, \min)$, quando este for viável, ou $\emptyset$ caso contrário.

BRANCH-BOUND-SIMPLIFICADO $(P, c, \min)$
    onde $(P, c, \min)$ é o programa linear relaxado
1 $y^* \leftarrow$ {Solução heurística para o problema, $\emptyset$ se não obteve solução}
2 $U \leftarrow \mathrm{val}(y^*)$
3 Ativos $\leftarrow \{P\}$
4 enquanto Ativos $\neq \emptyset$ faça
5   escolha $Q \in$ Ativos e remova Q de Ativos
6     $y \leftarrow$ OPT-LP$(Q, c, \min)$
7   se $val(y) < U$ então
8   se y é inteiro então
9             $U \leftarrow \mathrm{val}(y)$;
10             $y^* \leftarrow y$
11   senão
12   seja y_i uma variável fracionária em y e α seu valor
13             $Q' \leftarrow Q \cap \{x : x_i \leq \lfloor \alpha \rfloor\}$
14             $Q'' \leftarrow Q \cap \{x : x_i \geq \lceil \alpha \rceil\}$
15   Ativos $\leftarrow$ Ativos $\cup \{Q', Q''\}$
16 devolva y^*.

O método Branch & Bound não necessita que a árvore de enumeração seja binária nem que a ramificação seja nas variáveis, estas são estratégias comumente usadas nas várias implementações deste método. É importante que em cada ramificação o espaço de soluções representado no nó ramificado deve estar representado nos seus ramos. Por simplicidade, na descrição acima, inserimos também os nós inteiros no conjunto de ativos, mas na prática, sempre que um nó é viável e inteiro, atualizamos a melhor solução conhecida sem inserir no conjunto de ativos.

EXEMPLO 5.4 *Considere o programa linear inteiro, que consiste em encontrar x que*

$$\begin{array}{ll} \text{maximize} & 3x_1 + 4x_2 \\ \text{sujeito a} & \left\{ \begin{array}{rrcl} -5x_1 & +3x_2 & \leq & 3\,, \\ 2x_1 & +4x_2 & \leq & 17\,, \\ 10x_1 & +4x_2 & \leq & 45\,, \\ x_1 & & \geq & 0\,, \\ & x_2 & \geq & 0\,, \\ & x_1,\ x_2 & \in & \mathbb{Z}\,. \end{array} \right. \end{array}$$

Na figura 17 apresentamos a ramificação gerada por uma execução específica do algoritmo Branch & Bound neste programa linear. A seguir, detalhamos como esta ramificação foi gerada.

Inicialmente a árvore começa apenas com o nó raiz **A**, *contendo o programa linear relaxado e valor de solução ótima igual a* $20{,}5$. *O nó raiz é ativo e tem solução ótima com valor fracionária em x_1 ($x_1 = 3{,}5$).*

O nó ativo **A** *é removido e é ramificado em outros dois nós, um (nó* **B***) com a adição da restrição $x_1 \geq 4$ ($\lceil 3{,}5 \rceil = 4$) e outro (nó* **C***) com a adição da restrição $x_1 \leq 3$ ($\lfloor 3{,}5 \rfloor = 3$). Agora temos dois nós ativos, os nós* **B** *e* **C**.

Vamos supor que o próximo nó a ser removido do conjunto de ativos é o nó **C**. *A solução ótima do programa linear associado a este nó é fracionário em x_2. Assim, este nó é ramificado em outros dois nós ativos um com a restrição adicional $x_2 \geq 3$ (nó* **D***) e outro adicionando $x_2 \leq 2$ (nó* **E***). Neste momento, temos três nós ativos, os nós* **B**, **D** *e* **E**.

Vamos escolher agora o nó **E**. *Este é um nó inteiro, portanto, podemos guardar esta solução em y^*, com valor 17, a melhor encontrada até o momento. Com isso os atuais nós ativos são os nós* **B** *e* **D**.

Agora, vamos escolher o nó **B**. *Verificamos que este nó é viável e seu programa linear apresenta solução de valor 17. Este é o mesmo valor da melhor solução que conhecemos, e portanto neste ramo não há soluções de valor melhor que a já obtida. Neste caso, este nó é simplesmente podado. Agora o único nó ativo é o nó* **D**.

Removendo-se o nó **D** *do conjunto de ativos, observamos que a solução ótima deste nó tem o valor de x_1 fracionário. Com isso ramificamos em dois nós, um adicionando a restrição $x_1 \leq 2$ (nó* **F***) e outro adicionando a restrição $x_1 \geq 3$. Este último nó não foi representado pois trata se de um nó inviável. Desta vez o único nó ativo restante é o nó* **F**.

Nesta iteração, o nó **F** *é removido e verificamos que tem solução ótima com valor de x_2 fracionário. Assim, geramos dois nós ativos, um com a restrição $x_2 \geq 4$ (nó não*

representado por se tratar de nó inviável) e outro com a restrição $x_2 \leq 3$ (nó $\mathbf{G}$). Agora o único ativo é o nó $\mathbf{G}$.

Na última iteração, o nó $\mathbf{G}$ é removido e verificamos que tem solução inteira de valor 18, melhor que o valor anterior de y^*. Neste momento atualizamos y^* com a solução deste nó.

Como não temos mais nós ativos, podemos devolver a solução inteira ótima em y^* de valor 18.

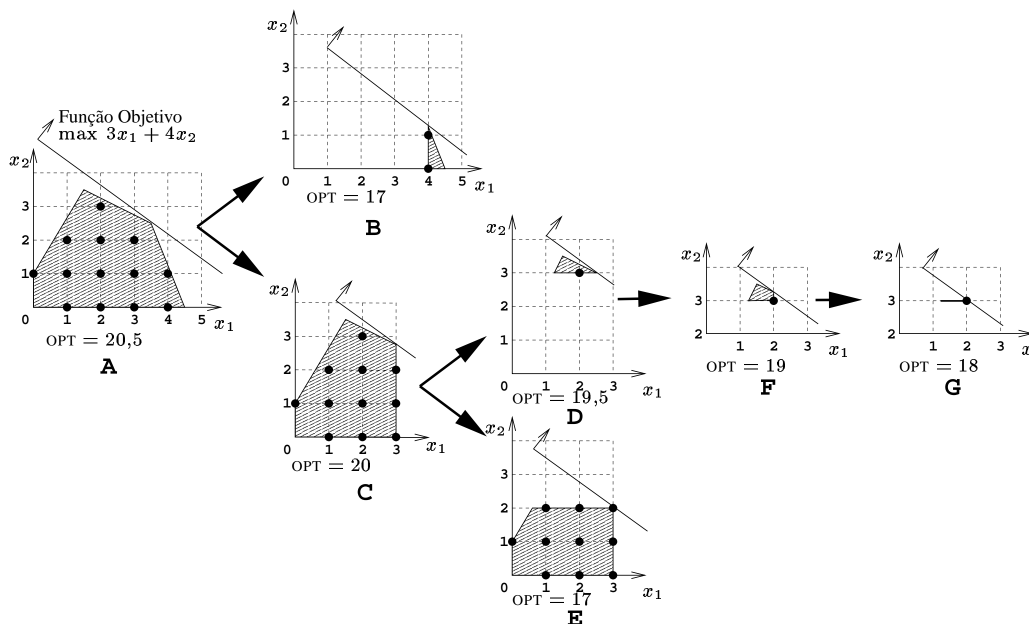


Figura 17: Uma enumeração da árvore de Branch & Bound, sem nós inviáveis.

Um caso particular deste tipo de Branch & Bound é quando as variáveis são binárias. Neste caso a ramificação de um nó simplesmente fixa uma variável em 0 em um dos ramos e no outro ramo a variável é fixada para 1.

Em geral, não é necessário se duplicar todo o programa linear durante a ramificação dos nós. Podemos simplesmente usar apenas um programa linear para toda a ramificação. Para obter a solução ótima do programa linear associado a um nó ativo, atualizamos as restrições particulares existentes neste nó. O conjunto de restrições particulares de um nó N pode ser obtido percorrendo as restrições que estão no caminho entre N e a raiz da árvore de Branch & Bound. Quando as restrições particulares do nó são restrições simples que apenas delimitam uma variável, em geral não é necessário se adicionar tal restrição, mas apenas atualizar os delimitantes da variável. Com isso é possível se chegar à uma nova solução mais rapidamente, pois os métodos para resolver programas lineares aproveitam parte das informações da solução anterior para o novo programa linear associado a este nó.

5.3.1. Estratégias envolvidas na árvore de Branch & Bound

A seguir, apresentamos outras estratégias envolvidas em implementações do método Branch & Bound.

1. Ao escolher um nó a ramificar, podemos escolher pelo nó que tem o pior delimitante da solução ótima. Por exemplo, em um problema de minimização, pegue o nó que tem o menor limite inferior. Isto permite diminuir a diferença entre os limitantes superior e inferior. Mas não é recomendado que se use apenas este tipo de escolha.

2. Na escolha da variável a ser ramificada, pode se escolher a variável "mais fracionária" (que tem a parte fracionária mais próxima de 0,5) ou a "menos fracionária" (que tem a parte fracionária mais distante de 0,5). No primeiro caso os ramos do nó podem apresentar maiores mudanças em relação a ele. No segundo, podemos chegar a soluções viáveis mais rapidamente.

3. Em vez de ramificação pelas variáveis, podemos ramificar através de restrições. Por exemplo, vamos supor que temos uma solução ótima x em um nó e suponha que o produto $ax = a_1x_1 + \ldots + a_nx_n$, deva ser inteiro para um certo vetor a, no entanto o produto ax nos dá um valor fracionário α. Neste caso, podemos ramificar o nó em outros dois, um com a restrição $ax \leq \lfloor\alpha\rfloor$ e outro com a restrição $ax \geq \lceil\alpha\rceil$.

4. Durante a busca, é interessante se usar heurísticas em alguns nós da árvore na tentativa de se obter soluções viáveis melhores.

5. Duas estratégias comuns para percorrer a árvore de ramificação, é a busca em profundidade e a busca em largura. Freqüentemente estas duas estratégias são combinadas para direcionar a busca em ramos promissores.

6. Em alguns problemas a existência de algumas variáveis fixadas pode implicar na fixação de outras variáveis através de implicações lógicas. Por exemplo, considere o problema do TSP resolvido através de Branch & Bound e programação linear. Dado um nó da árvore, considere o grafo obtido do original da seguinte forma: remova as arestas com a correspondente variável fixada em 0 e faça uma contração das arestas com variável fixada em 1. Se no grafo resultante existir vértice com exatamente duas arestas incidentes, podemos incluir estas arestas na solução do nó, fixando as variáveis correspondentes em 1.

5.4. Método de Planos de Corte

Na seção 3, vimos vários problemas para os quais o poliedro inteiro estava descrito através de poucas desigualdades. Isto também ocorreu para o problema de emparelhamento de peso máximo em grafos bipartidos. Estes são bons casos, em que pudemos resolver o problema eficientemente apenas com um programa linear. Por outro lado, para muitos problemas não temos uma descrição por programação linear através de poucas desigualdades (certamente isto não é esperado para nenhum problema $\mathcal{NP}$-difícil).

Note que em geral, não precisamos de uma descrição completa de um poliedro inteiro P. Em geral desejamos uma solução ótima $x^* \in P$, para uma certa função objetivo $f(x)$. Neste caso, a direção de busca da solução ótima pode ser guiada por esta função objetivo.

Suponha que não temos P, mas temos um poliedro limitado Q que é uma aproximação de P, onde $P \subseteq Q$. Seja y_0 uma solução ótima de Q para a função objetivo. Se $y_0 \in P$, devolvemos o ponto y_0 como solução ótima do problema. Caso $y_0 \notin P$, procuramos por uma desigualdade $ax \leq \gamma$ tal que $P \subseteq \{x : ax \leq \gamma\}$ e $ay_0 > \gamma$. I.e., todos os pontos de P satisfazem esta desigualdade, mas o ponto y_0 não satisfaz. Dizemos que esta desigualdade é um *plano de corte* que separa y_0 de P. Neste caso, podemos obter

um novo poliedro Q', adicionando esta desigualdade a Q. Em seguida, encontramos uma solução ótima de Q', digamos y_1, e verificamos se y_1 pertence ou não a P. No primeiro caso devolvemos y_1 como solução ótima, caso contrário repetimos a busca de um plano de corte que separa y_1. Este processo se repete até que uma solução ótima de P seja encontrada. A figura 18 ilustra uma seqüência de inserções de planos de corte, até que uma solução ótima em P seja encontrada.

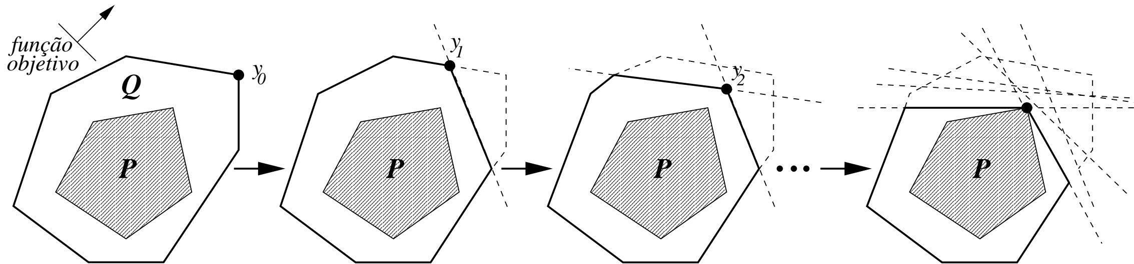


Figura 18: Seqüência de inserções de planos de corte.

Uma simplificação do método de planos de corte é apresentada a seguir.

MÉTODO DE PLANOS DE CORTE$(Q, c, \min)$
    onde $(Q, c, \min)$ é um programa linear inicial
1 $y \leftarrow$OPT-LP$(Q, c, \min)$
2 enquanto $y \neq \emptyset$ e encontra plano $ax \leq \gamma$ que separa y faça
3     $Q \leftarrow Q \cap \{x : ax \leq \gamma\}$
4     $y \leftarrow$OPT-LP$(Q, c, \min)$
5 devolva y.

O poliedro inicial Q pode começar com um poliedro limitado por poucas restrições, mas para evitar iterações excessivas, é conveniente começar Q com um bom conjunto de restrições.

Algumas perguntas que aparecem nesta estratégia são: *(i)* Como encontrar os planos de corte ? *(ii)* Quais os melhores planos de corte ? *(iii)* Quantas vezes o processo se repete ? A seguir, iremos discutir brevemente estas questões.

5.4.1. Otimização $\times$ Separação

Um resultado central nesta área é a equivalência entre o problema de separação e o problema de otimização. Enunciaremos este resultado de maneira simplificada. O leitor poderá encontrar mais informações sobre estes resultados em [Grötschel et al., 1981].

O problema da separação consiste no seguinte.

Problema DA SEPARAÇÃO: *Seja $P \subseteq \mathbb{R}^n$ um poliedro limitado e $y \in \mathbb{R}^n$, determinar se $y \in P$, caso contrário, encontrar desigualdade $ax \leq b$ tal que $P \subseteq \{x : ax \leq b\}$ e $ay > b$.*

O problema de otimização pode ser formulado como segue.

Problema DE OTIMIZAÇÃO: *Seja $P \subseteq \mathbb{R}^n$ um poliedro limitado e $c \in \mathbb{R}^n$, encontrar $x^* \in P$ tal que cx^* é máximo, ou mostre que P é vazio.*

Grötschel, Lovász e Schrijver [Grötschel et al., 1981] mostraram que a eficiência de um problema de otimização está intimamente ligado com o problema de separação deste problema.

TEOREMA 5.4 *Para qualquer classe de poliedros próprios, o problema de otimização pode ser resolvido em tempo polinomial se e somente se o problema da separação pode ser resolvido em tempo polinomial.*

Não definiremos a classe de poliedros próprios para não entrar em maiores tecnicalidades, mas a maioria dos problemas em otimização combinatória está associada a tais poliedros.

Para alguns problemas, é possível se resolver o problema de separação associado de maneira eficiente. Veremos um exemplo disto posteriormente.

5.4.2. Desigualdades Válidas, Faces e Facetas

Nesta seção resumiremos alguns tipos de desigualdades importantes da área. Os melhores planos de corte são aqueles que além de separar o ponto inválido, estão próximos do poliedro inteiro, ou chegam a encostar no poliedro. Com isso, podemos classificar as desigualdades em desigualdades válidas, faces e facetas.

Seja P um poliedro. Dizemos que $ax \leq b$ é uma *desigualdade válida* para P se $P \subseteq \{x : ax \leq b\}$. Um conjunto F é dito ser uma *face* de P se existe desigualdade válida $ax \leq b$ tal que $F = P \cap \{x : ax \leq b\}$. Uma face F de P é dita ser *própria* se $F \neq P$. Uma face F de P é dita ser *não-trivial* se $\emptyset \neq F \neq P$. Uma face não-trivial F é dita ser uma *faceta* de P se F não está contida em outra face própria de P. A figura 19 ilustra estas desigualdades em um poliedro.

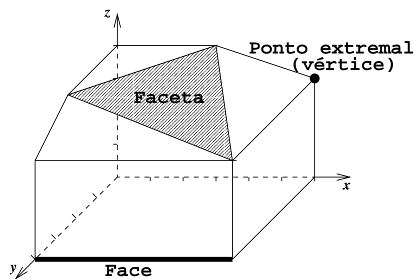


Figura 19: Desigualdades Válidas, Faces e Facetas.

Observe que as melhores desigualdades válidas são aquelas que induzem facetas.

5.4.3. Geração de Planos de Corte para programa linear inteiro

Gomory [Gomory, 1958] e Chvátal [Chvátal, 1973] foram os primeiros a desenvolver métodos automatizados e finitos para se obter planos de corte em problemas de programação linear inteiro. A seguir, veremos como é o método de cortes obtido por

Chvátal, que foi desenvolvido a partir do método de Gomory. Vamos chamá-los de *Cortes de Chvátal-Gomory*.

Suponha que queremos soluções inteiras, $x \in \mathbb{Z}^n$ para um poliedro $P := \{x : Ax \leq b\}$, $A \in \mathbb{Q}^{m \times n}$ e $b \in \mathbb{Q}^m$. A idéia do método consiste em combinar as linhas do sistema $Ax \leq b$ para obter uma desigualdade $ax \leq t$ tal que $\lfloor a \rfloor x \leq \lfloor t \rfloor$ é uma desigualdade válida. Após isto, a nova desigualdade $\lfloor a \rfloor x \leq \lfloor t \rfloor$ é inserida no sistema.

Uma desigualdade $ax \leq t$, sendo a um vetor inteiro, pertence ao fecho elementar de P, denotado por $e^1(P)$ se existem desigualdades $d_1 x \leq f_1, \ldots, d_n x \leq f_m$ de P e valores não negativos $\lambda_1, \ldots, \lambda_m$ tal que $\sum_{i=1}^{m} \lambda_i d_i = a$ e $\lfloor \sum_{i=1}^{m} \lambda_i f_i \rfloor \leq t$. Para $k > 1$, definimos $e^k(P) := e^1(P \cup e^{k-1}(P))$. O fecho $c(P)$ é definido como $\cup_{k=1}^{\infty} e^k(P)$. O seguinte teorema foi provado por Chvátal [Chvátal, 1973].

TEOREMA 5.5 *Se P é um poliedro limitado e I o conjunto de pontos inteiros em P, então conv(I) pode ser obtido após um número finito k de operações do fecho.*

A seguir, aplicamos o método de Chvátal-Gomory para obter cortes para o problema de emparelhamento de peso máximo em grafos.

5.4.4. Problema do Emparelhamento de Peso Máximo

Na seção 5.1.1 vimos que o problema do emparelhamento de peso máximo em grafos bipartidos pode ser bem resolvido apenas com programação linear, através da versão relaxada do programa linear dos emparelhamentos, reproduzida a seguir. Dado um grafo $G = (V, E)$, encontrar x que

$$\begin{array}{lll} & \text{maximize} & \sum_{e \in E} c_e x_e \\ (P) & \text{sujeito a} & \left\{ \begin{array}{rcll} \sum_{e \in \delta(v)} x_e & \leq & 1 & \forall v \in V, \\ x_e & \geq & 0 & \forall e \in E. \end{array} \right. \end{array}$$

Se P_{Emp} é o poliedro inteiro dos emparelhamentos e G é bipartido, pelo teorema 5.1, sabemos que $P = P_{Emp}$. Já para o caso de grafos que não são bipartidos, este resultado pode não ser válido. O exemplo 5.5 ilustra este caso.

EXEMPLO 5.5 *Considere o grafo da figura 20 com pesos unitários nas arestas. A atribuição $x = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ é a única solução ótima de (P) e tem valor 2,5.*

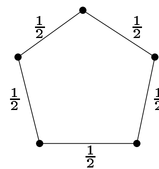


Figura 20: Solução ótima fracionária para a relaxação do programa linear dos emparelhamentos.

Vamos aplicar o método de Chvátal para obter cortes para (P). Seja $U \subseteq V$ tal que $|U| \geq 3$ é ímpar. Some as desigualdades $\sum_{e \in \delta(u)} x_e \leq 1$ para todo $u \in U$, obtendo a

desigualdade

$$2\sum_{e\in E(U)} x_e + \sum_{e\in\delta(u)} x_e \leq |U|,$$

onde $E(U)$ são arestas com ambos extremos em U. Ignorando o segundo termo, temos a desigualdade $\sum_{e\in E(U)} x_e \leq \frac{|U|}{2}$. Como o lado esquerdo da desigualdade deve ser inteiro e o direito é fracionário, podemos arredondar para baixo o lado direito, obtendo $\sum_{e\in E(U)} x_e \leq \lfloor\frac{|U|}{2}\rfloor$. Assim, a nova desigualdade válida para este problema é a seguinte:

$$\sum_{e\in E(U)} x_e \leq \frac{|U|-1}{2}, \qquad U \subseteq V, \quad |U| \geq 3 \text{ ímpar.}$$

Considere o novo programa linear (P_{Edm}) com os cortes obtidos desta forma: Encontrar x que

$$\begin{array}{lll} \text{maximize} & \sum_{e\in E} c_e x_e & \\ (P_{Edm}) \text{ sujeito a} & \left\{ \begin{array}{rcll} \sum_{e\in\delta(v)} x_e & \leq & 1 & \forall v \in V, \\ \sum_{e\in E[S]} x_e & \leq & \frac{|S|-1}{2} & \forall S \subseteq V, \ |S| \text{ ímpar}, \\ x_e & \geq & 0 & \forall e \in E\,, \end{array} \right. \end{array}$$

onde $E[S] := \{e \in E : e \text{ tem ambos extremos em } S\}$.

O seguinte teorema, provado por Edmonds [Edmonds, 1965], mostra que se inserirmos todas as desigualdades representadas no programa (P_{Edm}), obtemos o poliedro inteiro dos emparelhamentos.

TEOREMA 5.6 $P_{Edm}(G) = P_{Emp}(G)$.

Apesar do programa (P_{Edm}) possuir uma quantidade exponencial de desigualdades, Padberg e Rao [Padberg and Rao, 1982], mostraram que o problema da separação das desigualdades de (P_{Edm}) pode ser resolvido por um algoritmo de tempo polinomial. Este resultado junto com o teorema 5.4, mostram que o problema de emparelhamento de peso máximo em grafos quaisquer pode ser resolvido eficientemente.

TEOREMA 5.7 *O problema do emparelhamento de peso máximo em grafos quaisquer pode ser resolvido em tempo polinomial.*

Apesar destes métodos para obter planos de corte serem métodos automatizados e aplicados na resolução de qualquer problema de programação linear inteira, em tempo finito, tais métodos são melhor aproveitados na busca de classes de desigualdades, mas computacionalmente não são em geral bem sucedidos. Em geral estes métodos são lentos e apresentam pequena melhora a cada inserção de plano de corte.

Uma estratégia que tem dado resultados melhores, é a inserção de planos de corte obtidos a partir das propriedades estruturais das soluções de cada problema, se possível de desigualdades que induzem facetas. Esta estratégia nos leva ao desenvolvimento de algoritmos específicos para um problema, por outro lado, a qualidade dos planos de

corte é melhor. Os primeiros a usar esta estratégia com sucesso foram Dantzig *et al.* [Dantzig et al., 1954] para o problema do TSP.

A seguir, veremos uma classe de planos de corte que aparecem em vários problemas de otimização envolvendo conectividade.

5.4.5. Planos de corte e Conectividade

Alguns dos problemas que vimos anteriormente, como o problema da árvore de Steiner e o problema do Caixeiro Viajante, envolvem a busca de soluções representadas por grafos conexos com algumas propriedades de conexidade.

Por exemplo, considere a formulação inteira do problema do caixeiro viajante (8). Duas dificuldades desta formulação são as seguintes: *(i)* é um problema de programação linear inteiro; *(ii)* tem número exponencial de desigualdades.

O número exponencial de desigualdades de restrição de corte,

$$\sum_{e \in \delta(S)} x_e \geq 2, \qquad \forall \emptyset \neq S \subset V,$$

do programa linear relaxado já nos traz dificuldades para obter uma solução ótima fracionária. Por outro lado, o problema de separação para este tipo de desigualdade pode ser resolvido em tempo polinomial. Isto implica que podemos pelo menos encontrar a solução ótima fracionária da relaxação do programa (8) eficientemente.

Para saber se um ponto x satisfaz a todas as desigualdades de corte, podemos testar se algum par de vértices, s e t, está separado por um corte $(S, V \setminus S)$ tal que

$$\sum_{e \in \delta(S)} x_e < 2 \quad s \in S, \quad t \in V \setminus S.$$

Para isto, podemos executar um algoritmo para encontrar um st-corte mínimo, onde cada aresta e tem capacidade x_e. Se o valor do st-corte mínimo for menor que 2, então o corte gerado viola a desigualdade de corte e podemos inseri-la como desigualdade válida.

Árvore de Cortes de Gomory-Hu:

Uma outra alternativa para encontrar cortes violados, é calcular o corte mínimo para todos os pares de vértices e inserir as desigualdades necessárias apenas para os cortes violados. Isto pode ser feito de maneira rápida através da árvore de cortes de Gomory-Hu, que faz uso de apenas $n - 1$ chamadas do algoritmo para st-corte mínimo. Cada aresta e de uma árvore de cortes T, representa um corte mínimo formado pelas arestas do grafo original que ligam as partes de $T - e$. O corte mínimo entre dois vértices é dado por uma aresta no caminho da árvore de corte que liga estes vértices e tem menor capacidade.

Na figura 21, apresentamos um grafo com *(a)* capacidades nas arestas, *(b)* uma árvore de cortes de Gomory-Hu para este grafo e *(c)* o corte mínimo separando os vértices A e G de capacidade 7 (7 é o mínimo em $\{11, 7, 10\}$).

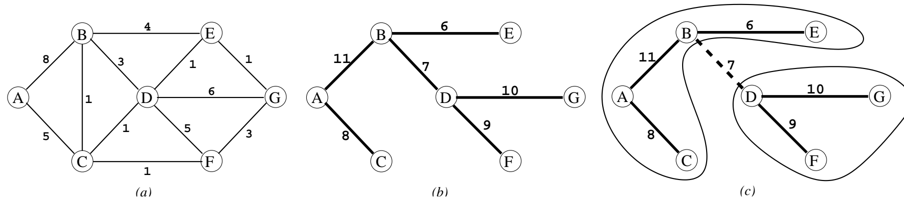


Figura 21: (a) Grafo com custos nas arestas. (b) Árvore de Gomory-Hu. (c) Corte mínimo entre A e G.

5.4.6. Exemplo de inserção de planos de corte para o TSP

Vamos ver um exemplo de como se comporta a inserção de cada desigualdade no problema do Caixeiro Viajante, para o grafo apresentado na figura 22.

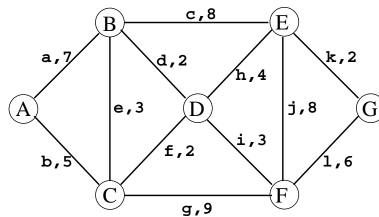
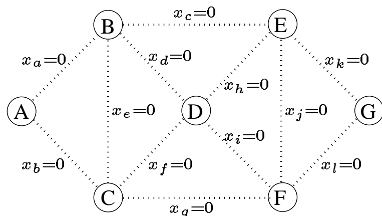


Figura 22: Grafo com nome de arestas e respectivos custos.

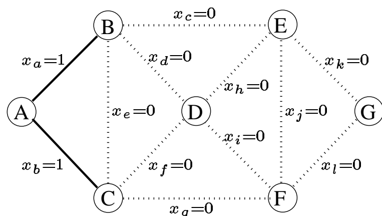
Para não sobrecarregar as figuras, em cada passo apresentaremos apenas os valores obtidos da solução ótima fracionária sem colocar os custos das arestas e seus nomes. Na próxima figura, apresentamos o programa linear sem nenhuma restrição, exceto os limitantes das variáveis na relaxação $(0 \leq x_i \leq 1)$. Neste caso, a solução ótima é o vetor nulo. Nas próximas iterações verificamos o comportamento do programa linear após a inserção de cada desigualdade.



$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \left\{ \; 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1. \right.$$

Solução ótima = 0.

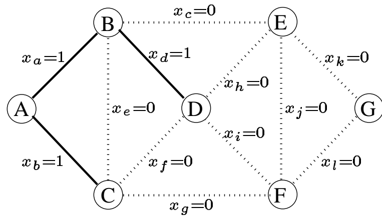
Apenas restrições $0 \leq x_e \leq 1, \; \forall e \in E.$



$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \left\{ \begin{array}{rcl} x_a + x_b & = & 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1. & & \end{array} \right.$$

Solução ótima = 12.

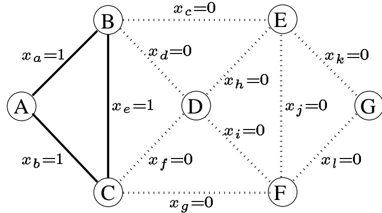
Após adicionar $x(\delta(A)) = 2.$



Após adicionar $x(\delta(B)) = 2$.

$$\text{min} \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b = 2, \\ x_a + x_c + x_d + x_e = 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1, \end{cases}$$

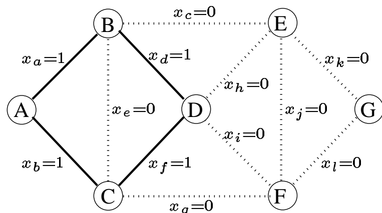
Solução ótima = 14.



Após adicionar $x(\delta(C)) = 2$.

$$\text{min} \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b = 2, \\ x_a + x_c + x_d + x_e = 2, \\ x_b + x_e + x_f + x_g = 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1. \end{cases}$$

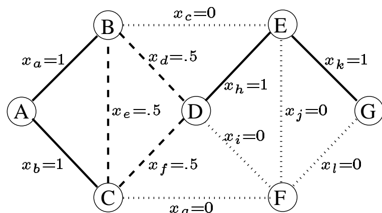
Solução ótima = 15.



Após adicionar $x(\delta(D)) = 2$.

$$\text{min} \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b = 2, \\ x_a + x_c + x_d + x_e = 2, \\ x_b + x_e + x_f + x_g = 2, \\ x_d + x_f + x_h + x_i = 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1. \end{cases}$$

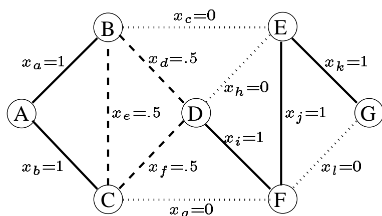
Solução ótima = 16.



Após adicionar $x(\delta(E)) = 2$.

$$\text{min} \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b = 2, \\ x_a + x_c + x_d + x_e = 2, \\ x_b + x_e + x_f + x_g = 2, \\ x_d + x_f + x_h + x_i = 2, \\ x_c + x_h + x_j + x_k = 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1, \end{cases}$$

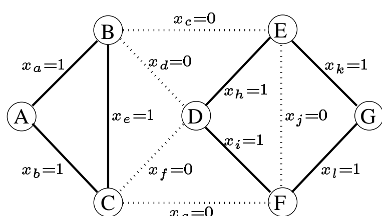
Solução ótima = 21,5.



Após adicionar $x(\delta(F)) = 2$.

$$\text{min} \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b = 2, \\ x_a + x_c + x_d + x_e = 2, \\ x_b + x_e + x_f + x_g = 2, \\ x_d + x_f + x_h + x_i = 2, \\ x_c + x_h + x_j + x_k = 2, \\ x_g + x_i + x_j + x_l = 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1. \end{cases}$$

Solução ótima = 28,5.

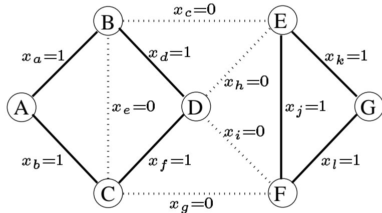


Após adicionar $x(\delta(G)) = 2$.

$$\text{min} \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b = 2, \\ x_a + x_c + x_d + x_e = 2, \\ x_b + x_e + x_f + x_g = 2, \\ x_d + x_f + x_h + x_i = 2, \\ x_c + x_h + x_j + x_k = 2, \\ x_g + x_i + x_j + x_l = 2, \\ x_k + x_l = 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1. \end{cases}$$

Solução ótima = 30.

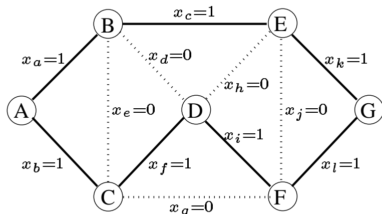
Todas as restrições $x(\delta(v)) = 2$ para todo vértice v foram inseridas. Agora vamos inserir as desigualdades de restrição de corte:



Após adicionar desigualdade do corte mínimo que separa A e D (corte de valor 0) $x(\delta(S)) \geq 2$, para $S = \{A, B, C\}$.

$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b &= 2, \\ x_a + x_c + x_d + x_e &= 2, \\ x_b + x_e + x_f + x_g &= 2, \\ x_d + x_f + x_h + x_i &= 2, \\ x_c + x_h + x_j + x_k &= 2, \\ x_g + x_i + x_j + x_l &= 2, \\ x_k + x_l &= 2, \\ x_c + x_d + x_f + x_g &\geq 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1. \end{cases}$$

Solução ótima = 32.



Após adicionar desigualdade do corte mínimo que separa A e E (corte de valor 0) $x(\delta(S)) \geq 2$, para $S = \{A, B, C, D\}$.

$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b &= 2, \\ x_a + x_c + x_d + x_e &= 2, \\ x_b + x_e + x_f + x_g &= 2, \\ x_d + x_f + x_h + x_i &= 2, \\ x_c + x_h + x_j + x_k &= 2, \\ x_g + x_i + x_j + x_l &= 2, \\ x_k + x_l &= 2, \\ x_c + x_d + x_f + x_g &\geq 2, \\ x_c + x_g + x_h + x_i &\geq 2, \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1. \end{cases}$$

Solução ótima = 33.

Ao fim da inserção de planos de corte, obtivemos uma solução inteira ótima para o TSP, de valor 33. Apesar do sucesso ao obter uma solução ótima inteira, em geral as inserções apenas de restrições de corte não são suficientes para resolver o problema. A seguir, apresentamos um exemplo onde a inserção de planos de corte de grau e de corte não são suficientes para obter uma solução inteira.

EXEMPLO 5.6 *Na figura 23, temos (a) um grafo com custos nas arestas e (b) uma solução ótima do programa relaxado do TSP.*

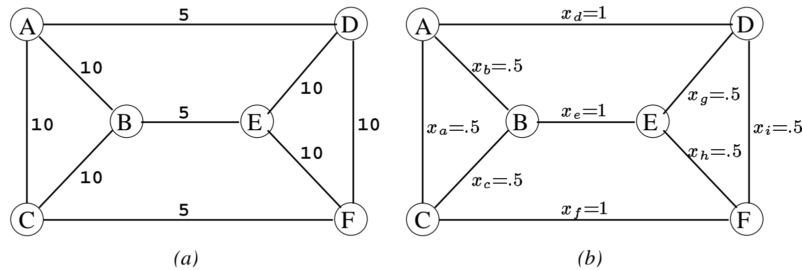


Figura 23: (a) Grafo com custos nas arestas. (b) Solução ótima fracionária.

A solução fracionária apresentado na figura 23-(b) pode ser separada através de *Comb Inequalities* [Edmonds, 1965, Chvátal, 1975, Grötschel and Padberg, 1979]. Na próxima seção veremos um método para continuar a busca de uma solução quando não sabemos separar a atribuição fracionária.

5.5. Branch & Cut

Um algoritmo Branch & Cut é uma combinação do método Branch & Bound e da aplicação do método de planos de corte durante a geração da árvore de Branch & Bound. Nesta estratégia investimos um grande esforço nos nós da árvore para limitar seu crescimento. Este esforço pode ser apresentado de várias maneiras, como o uso de estratégias

de separação, heurísticas primais em cada nó, métodos de ramificação, pré-processamento em cada nó, gerenciamento de desigualdades válidas, entre outras. Muitas destas estratégias podem ser aplicadas aos métodos anteriores, mas são estratégias que combinadas no método Branch & Cut têm obtido sucesso para muitos problemas de otimização combinatória.

A seguir, apresentamos uma descrição simplificada do método Branch & Cut para um problema de minimização. Usaremos a mesma notação usada no algoritmo Branch & Bound.

BRANCH-CUT-SIMPLIFICADO $(P, c, \min)$
onde $(P, c, \min)$ é um programa linear inicial

```
1   y* ← {Solução heurística para o problema, ∅ se não obteve solução}
2   U ← val(y*)
3   Ativos ← {P}
4   enquanto Ativos ≠ ∅ faça
5       escolha Q ∈ Ativos e remova Q de Ativos
6       y ← OPT-LP(Q, c, min)
7       enquanto y ≠ ∅ e encontra plano ax ≤ γ que separa y faça
8           Q ← Q ∩ {x : ax ≤ γ}
9           y ← OPT-LP(Q, c, min)
10      se val(y) < U então
11          se y é inteiro então
12              U ← val(y);
13              y* ← y
14          senão
15              seja y_i uma variável fracionária em y e α seu valor
16              Q′ ← Q ∩ {x : x_i ≤ ⌊α⌋}
17              Q″ ← Q ∩ {x : x_i ≥ ⌈α⌉}
18              Ativos ← Ativos ∪ {Q′, Q″}
19  devolva y*.
```

5.5.1. Pré-processamento

Antes de começar a atacar intensivamente um problema é conveniente fazer um pré-processamento de maneira a diminuir o tamanho do problema.

Uma das estratégias para se diminuir o tamanho do problema é através da fixação de variáveis por custo reduzido. O custo reduzido de uma variável não básica indica a mudança no valor da solução ótima caso esta venha fazer parte de uma solução básica do programa linear. A maioria dos pacotes para resolver programas lineares apresenta rotinas para verificar se uma variável é não básica e obter seu custo reduzido.

Por exemplo, considere um problema de minimização de um programa linear binário P_I e suponha que temos uma solução (inteira) viável para este programa, de valor U. Seja P o programa linear relaxado de P_I e $x = (x_1, \ldots, x_n)$ uma solução ótima (fracionária) de P. Denote por L o valor da solução x. Seja x_i uma variável não básica e

denote por r_i seu custo reduzido. Seja t o valor da variável x_i, que pode ser igual a 0 ou 1. Se $|r_i| + L \geq U$, então podemos fixar a variável x_i em t. Isto ocorre pois a inclusão da variável x_i na base, com valor trocado, nos leva a uma solução com valor maior ou igual ao da solução viável existente. Assim, a variável x_i pode ser fixada sem perda da solução ótima.

Uma vez que algumas variáveis foram fixadas, possivelmente podemos, através de outras estratégias de pré-processamento fixar mais variáveis, como pela fixação de variáveis por implicação lógica, já comentado no método Branch & Bound. Após isto, é possível inclusive repetir os pré-processamentos anteriores, até que não possamos mais diminuir o tamanho do problema.

Este método é em geral bastante efetivo quando os limitantes superior e inferior estão próximos, ou quando temos uma variação grande nos custos da função objetivo.

Estas estratégias podem ser também aplicadas em cada nó da árvore de Branch & Cut, embora neste caso o investimento deva ser feito com cautela, para evitar investimento computacional desnecessário, pois o resultado do pré-processamento em um nó pode ser válido apenas para o nó e seus descendentes.

5.5.2. Pool de desigualdades

Como o custo para gerar desigualdades é em geral alto, convém guardar as restrições obtidas dentro de um depósito de desigualdades, *pool de desigualdades*. Quando buscamos planos de corte para um nó, podemos percorrer as desigualdades do pool e inserir aquelas que são válidas para o nó, evitando gerar novas desigualdades. Novos planos de corte são gerados se esta busca no pool é infrutífera.

Como o pool de desigualdades pode crescer bastante, contendo muitas desigualdades desnecessárias, convém manter uma *idade* para cada desigualdade. Cada vez que o pool é percorrido, aumenta se a idade das desigualdades desnecessárias. Assim, periodicamente, eliminamos aquelas desigualdades desnecessárias com certa idade mínima.

5.5.3. Estratégias envolvidas na árvore de Branch & Cut

A seguir, apresentamos outras estratégias envolvidas em implementações do método Branch & Cut. Neste caso, continuam valendo as estratégias e considerações já descritas para o método Branch & Bound.

1. As vezes vale mais a pena usar uma heurística para obter planos de corte rapidamente, que algoritmos que garantidamente determinam a existência de classe de planos de corte, mas a um alto custo computacional.

2. *Tailing off:* A inserção de desigualdade de separação de alguns pontos pode gerar melhorias pequenas e sobrecarregar muito o sistema. Neste caso, pode ser mais conveniente não separar estes pontos e ir direto para a ramificação.

3. É em geral mais efetivo escolher desigualdades de diferentes classes, que concentrar desigualdades de mesma classe.

4. Escolha as desigualdades mais violadas. Se $a \cdot x \leq \alpha$ é a desigualdade encontrada, calcule $a \cdot x' - \alpha$, onde x' é a solução ótima da relaxação representada no nó. Quanto maior for a diferença, maior a chance de se diminuir a distância do limitante superior com o inferior.

5. Selecione desigualdades que cobrem todo espaço de variáveis. Um sistema linear "se adapta" a cada nova desigualdade, mudando o mínimo. Quando introduzimos desigualdades com grande quantidade de variáveis não nulas, a chance disso ocorrer é menor.

6. Periodicamente, aplique heurísticas primais nos nós da árvore de Branch & Cut.

7. Em vez de buscar soluções ótimas, pare o algoritmo após um certo tempo de execução. Neste caso, devolva também a qualidade da melhor solução obtida. Por exemplo, se o problema é de minimização, devolva a melhor solução, digamos de valor U e o valor L, correspondente ao menor valor entre os nós ativos restantes. Neste caso, a solução obtida é no pior caso U/L vezes o valor da solução ótima do problema.

6. Conclusão

Esperamos que ao fim deste texto o leitor esteja mais motivado a entender a área de Otimização Combinatória de forma mais aprofundada e aprender outras técnicas e métodos que não pudemos apresentar por falta de espaço.

Referências

Birkhoff, G. (1946). Tres observaciones sobre el algebra lineal. *Revista Facultad de Ciencias Exactas, Puras y Aplicadas*, 5:147–151.

Bondy, J. and Murty, U. (1976). *Graph Theory with Applications*. Macmillan/Elsevier.

Carvalho, M., Cerioli, M., Dahab, R., Feofiloff, P., Fernandes, C., Ferreira, C., Guimarães, K., Miyazawa, F., Jr., J. P., Soares, J., and Wakabayashi, Y. (2001). *Uma introdução sucinta a algoritmos de aproximação*. Editora do IMPA - Instituto de Matemática Pura e Aplicada, Rio de Janeiro–RJ. M.R. Cerioli, P. Feofiloff, C.G. Fernandes, F.K. Miyazawa (editors).

Chvátal, V. (1975). On certain polytopes associated with graphs. *Journal of Combinatorial Theory B*, 18:138–154.

Chvátal, V. (1983). *Linear Programming*. Freeman.

Chvátal, V. (1973). Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:185–224.

Cook, S. (1971). The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, second edition.

Dantzig, G. (1951). Maximization of a linear function of variables subject to linear inequalities. In Koopmans, C., editor, *Activity Analysis of Production and Allocation*, pages 339–347. Wiley, New York.

Dantzig, G. (1963). *Linear Programming and Extensions*. Princeton University Press.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling salesman problem. *Operations Research*, 2:393–410.

Edmonds, J. (1965). Maximum matching and a polyhedron with $0, 1$-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130.

Ferreira, C. and Wakabayashi, Y. (1996). *Combinatória Poliédrica e Planos-de-Corte Faciais*. 10ª Escola de Computação, Campinas.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability, A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, CA.

Goemans, M. and Williamson, D. (1994). New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666.

Gomory, R. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278.

Grötschel, M. and Padberg, M. W. (1979). On the symmetric travelling salesman problem I: Inequalities. *Mathematical Programming*, 16:265–280.

Grötschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197.

Jain, K. (2001). A factor 2 approximation algorithm for the generalized Steiner network design problem. *Combinatorica*, 21:39–60.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395.

Karp, R. (1972). Reducibility among combinatorial problems. In Miller, R. and Thatcher, J., editors, *Complexity of Computer Computations*, pages 85–103. Plenum.

Khachiyan, L. (1979). A polynomial algorithm for linear programming. *Soviet Mathematical Doklady*, 20:191–194.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. John Wiley and Sons, New York.

Padberg, M. and Rao, M. (1982). Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 1:67–80.

Papadimitriou, C. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall.

Raghavan, P. and Thompson, C. (1987). Randomized rounding. *Combinatorica*, 7:365–374.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics. John Wiley, Chichester.

Vazirani, V. (2000). *Approximation Algorithms*. Springer-Verlag.

von Neumann, J. (1953). A certain zero sum two persons game equivalent to the optimal assignment problem. In Kuhn, H. and Tucker, A., editors, *Contributions to the Theory of Games II*, volume 28, pages 5–12.

Wolsey, L. (1998). *Integer Programming*. Wiley.