4
Resolução de *IPs*

A teoria de programação linear foi proposta na década de 40 e logo foi observado que seria desejável a resolução de problemas que apresentavam variáveis do tipo inteiro [37]. Isto levou ao desenvolvimento dos primeiros algoritmos que utilizam a idéia de se adicionar novas desigualdades à formulação do problema, os chamados *cutting plane algorithms*, propostos por Dantzig, Fulkerson e Johnson [34] e Gomory [57, 58, 59]. Em sequência, Land e Doig introduziram em 1960 um algoritmo enumerativo que utiliza estimativas no valor da solução ótima para evitar a enumeração de todas as soluções possíveis (algoritmo *branch-and-bound* [71]). A partir de então abordagens como enumeração implícita ([15]), decomposição (Benders [21]), relaxação lagrangeana (Geoffrion [52]) e diversas heurísticas (veja Zanakis e Evans [119]) foram propostas para a resolução de *IPs*.

Infelizmente, hoje após mais de 50 anos de pesquisas nessa área, nenhum dos algoritmos disponíveis mostrou-se adequado para resolver satisfatoriamente toda e qualquer instância de problemas de programação linear inteira. Atualmente, boa parte dos esforços de pesquisa concentram-se no desenvolvimento de técnicas e algoritmos para classes particulares de problemas. Muitos desses algoritmos são variações do algoritmo *branch-and-bound*, descrito na seção 4.1), adaptado a algum problema em particular.

Neste capítulo, que trata da resolução de *IPs* por algoritmos enumerativos, inicialmente é descrito na seção 4.1 o algoritmo *branch-and-bound* propriamente dito. Nas seções 4.2, 4.3 e 4.4 são apresentadas breves descrições gerais das variações *branch-and-cut*, *branch-and-price* e *branch-and-cut-and-price*, respectivamente. Descrições mais detalhadas dessas variações podem ser encontradas nas referências [95, 70, 113, 84, 16], que foram a base para o texto dessas seções.

Para simplificar as descrições do algoritmo *branch-and-bound* e de suas variações, estas serão concentradas em formulações de programação linear inteira que não contém variáveis contínuas, ou seja, em *IPs*. Seja P um tal problema de programação linear inteira definido por 2-2. Essas descrições

utilizam, como estimativa do valor da solução ótima de P, o valor da solução ótima de um problema de programação linear, habitualmente referido por relaxação linear de P e definido como $\overline{z} = min\{f(x) = c.x \mid x \in \overline{\mathcal{F}} \supseteq \mathcal{F}\}$, onde

$$\overline{\mathcal{F}} = \{x \in \mathbb{R}^n_+ \mid A.x \geq b\}\,. \tag{4-1}$$

4.1 *Branch-and-Bound*

O método *Branch-and-bound*, proposto por Land and Doig [71], para a resolução de problemas de programação inteira utiliza uma estratégia de divisão e conquista. O seu princípio básico é o uso de estimativas no valor da solução ótima de um problema, para evitar a inspeção de partes de seu conjunto de soluções.

Seja P um problema de programação linear inteira, definido por 2-2, e $\mathcal{F}$, definido por 2-3, o seu conjunto de soluções. Considere a relaxação linear $\overline{P}$ de P (definição 2-6) e o seu conjunto de soluções que é o poliedro $\overline{\mathcal{F}}$ (definição 2-7).

Um *branching* consiste na definição de k poliedros $\overline{\mathcal{F}}_1, \ldots, \overline{\mathcal{F}}_k$, todos contidos em $\overline{\mathcal{F}}$, tais que $\bigcup_{i=1}^{k} (\overline{\mathcal{F}}_i \cap \mathbb{Z}^n_+) = \mathcal{F}$. Isso implica que o problema P no poliedro original pode ser resolvido através da resolução de k subproblemas inteiros em poliedros menores.

A operação de *bounding* consiste em encontrar a melhor solução contida em cada poliedro $\overline{\mathcal{F}}_i$ que pode ser obtido através da operação de *branching*. Esta operação fornece um modo de se calcular uma estimativa inferior para o valor de qualquer solução factível para o problema P em $\overline{\mathcal{F}}_i$, uma vez que para todo $\overline{\mathcal{F}}_i \subset \overline{\mathcal{F}}$ tem-se:

$$\min\{f(x) = c.x \mid x \in \overline{\mathcal{F}}_i\} \leq \min\{f(x) = c.x \mid x \in \mathcal{F} \cap \overline{\mathcal{F}}_i\}, \tag{4-2}$$

O algoritmo *branch-and-bound* trabalha em iterações, mantendo uma coleção L de subconjuntos de $\overline{\mathcal{F}}$, a melhor solução factível conhecida $x^* \in \mathcal{F}$ e o seu valor $z^* = f(x^*) = c.x^*$. Inicialmente $L = \{\overline{\mathcal{F}}\}$ e $z^* = \infty$. Assim, ao término do algoritmo, se o problema P é factível, x^* será uma solução ótima para o mesmo.

Na iteração i do algoritmo, um subconjunto $\overline{\mathcal{F}}_i$ é extraído de L e resolvida a relaxação a seguir:

$$\overline{z}_i = \min\{f(x) = c.x \mid x \in \overline{\mathcal{F}}_i\}. \tag{4-3}$$

Se o problema 4-3 é infactível ou se $\overline{z}_i \leq z^*$, então não existe nenhuma solução factível $x \in \mathcal{F} \cap \overline{\mathcal{F}}_i$ cujo valor da função objetivo seja menor do que o da melhor solução conhecida. Neste caso, o algoritmo prossegue com a iteração $i+1$. Caso contrário, se a solução ótima $\overline{x}_i$ do problema 4-3 é solução factível de P e $\overline{z}_i < z^*$, então faz-se $x^* = \overline{x}_i$ e $z^* = \overline{z}_i$. Além disso, todos os subconjuntos $\overline{\mathcal{F}}_j \in L$ tais que $\overline{z}_j \geq z^*$ são excluídos de L. Caso a solução ótima $\overline{x}_i$ do problema 4-3 não seja solução factível de P, é feito um *branching* no conjunto $\overline{\mathcal{F}}_i$, os subconjuntos resultantes são inseridos em L e o algoritmo prossegue com a iteração $i+1$. O algoritmo termina quando L torna-se vazio. Ao término, o problema P é infactível se $z^* = \infty$, senão z^* é o valor da solução ótima.

Além desse esquema básico, pode-se usar uma heurística para encontrar uma solução inicial x^* de valor z^* na primeira iteração ($i = 0$), o que pode diminuir o número de iterações do algoritmo. Outra opção é utilizar uma heurística que transforme uma solução $\overline{x} \in \overline{\mathcal{F}}$ em uma solução $x \in \mathcal{F}$. Assim, a cada iteração i, em que $\overline{x}_i \notin \mathcal{F}$, essa heurística seria utilizada para tentar melhorar o valor da estimativa z^*.

Cada execução do algoritmo *branch-and-bound* pode ser associada a uma árvore orientada T. Seja N a última iteração de uma execução finita do algoritmo. A cada iteração i é associado um vértice v_i em T ($i \in \{1, \cdots, N\}$). O vértice v_1 é a raiz da árvore. Para cada iteração $j > 1$ é adicionado o arco $\{v_i, v_j\}$, para cada v_j corresponde a um subconjunto resultante do *branching* a partir de $\overline{\mathcal{F}}_i$. A árvore $T = (V, E)$, com raiz v_1, é definida por $V = \{v_i \,|\, i \in \{1, \cdots, N\}\}$ e $E = \{\{v_i, v_j\} \,|\, \overline{\mathcal{F}}_j \in \{\overline{\mathcal{F}}_i^1, \cdots, \overline{\mathcal{F}}_i^k\}$ e $\overline{\mathcal{F}}_i = \bigcup_{r=1}^{k} \overline{\mathcal{F}}_j^r\}$

O esquema de enumeração, ou seja, o critério de escolha do subconjunto $\overline{\mathcal{F}}_i \in L$ a cada iteração i e o esquema de *branching* de $\overline{\mathcal{F}}_i$ são determinantes no tamanho da árvore T e, portanto, no tempo de execução do algoritmo. A escolha desses esquemas em geral são ditados pela experiência e conhecimento a respeito do problema a ser resolvido.

O esquema de *branching*, em particular, quase sempre depende da estrutura do problema em questão. Um *branching* bastante utilizado particiona o conjunto de soluções em dois através da imposição da restrição $x_j \leq \lfloor \overline{x}_j \rfloor$, para o primeiro, e $x_j \geq \lceil \overline{x}_j \rceil$, para o segundo, onde $\overline{x}_j$ é uma variável de valor fracionário da solução do LP corrente. Esquemas mais complexos, ou que geram mais de dois subconjuntos, são descritos em [16].

Embora o critério de escolha do subconjunto $\overline{\mathcal{F}}_i \in L$ a cada iteração i também possa depender da estrutura do problema, existem alguns esquemas genéricos que freqüentemente são utilizados. Esses critérios, ou esquemas de enumeração, dizem respeito à forma de se percorrer a árvore de enumeração.

A busca em profundidade (*depth-first*) escolhe o subconjunto incluído mais recentemente na lista L, a busca em largura (*breadth-first*) escolhe sempre o mais antigo na lista L e o *best-first* escolhe o subconjunto que fornece a melhor estimativa do valor da solução, ou seja, escolhe o subconjunto $\overline{\mathcal{F}}_i$ tal que o valor $\overline{z}_i$, dado por 4-3, seja mínimo.

4.2 *Branch-and-Cut*

O algoritmo *branch-and-bound* (veja seção 4.1) pode ser refinado para, dado um problema P de programação inteira (definido por 2-2), resolver relaxações lineares correspondentes a formulações de P com um número grande (potencialmente exponencial) de restrições. Tais formulações são de grande interesse devido ao fato de serem as únicas possíveis para determinados problemas ou, em outros casos, fornecerem melhores estimativas para o valor da solução ótima do problema do que formulações com um número polinomial de restrições. Este refinamento é conhecido como *branch-and-cut* [87].

Inicialmente, seja $\mathcal{F}$, definido por 2-3, o conjunto de soluções do problema P. Seja, ainda, $\overline{P}$ a relaxação linear de P, definição 2-6, e $\overline{\mathcal{F}}$ o conjunto de soluções de $\overline{P}$ (definição 2-7).

Considere o par $(\pi^T, \pi_0) \in \mathbb{R}^n \times \mathbb{R}$. A desigualdade $\pi^T.x \leq \pi_0$ é válida para o poliedro $\overline{\mathcal{F}}$, se $\overline{\mathcal{F}} \subseteq \{x \in \mathbb{R}^n_+ \mid \pi^T.x \leq \pi_0\}$. O conjunto $F = \overline{\mathcal{F}} \cap \{x \in \mathbb{R}^n_+ \mid \pi^T.x = \pi_0\}$ é uma face de $\overline{\mathcal{F}}$ definida pela desigualdade válida $\pi^T.x \leq \pi_0$. F é uma face própria de $\overline{\mathcal{F}}$ se $F \neq \emptyset$ e $F \neq \overline{\mathcal{F}}$. Uma face própria é chamada de faceta se não está contida em qualquer outra face própria de $\overline{\mathcal{F}}$. Dentre o conjunto de desigualdades válidas, aquelas que definem facetas são as mais fortes, uma vez que não podem ser redundantes (veja [118] ou [85]).

Seja, agora, $\overline{\Pi} \subseteq \mathbb{R}^n \times \mathbb{R}$ uma família de desigualdades válidas para o envoltório convexo ($conv(\overline{\mathcal{F}})$) de $\overline{P}$. O conjunto $\overline{\Pi}$ contém os planos de cortes que podem potencialmente ser usados para limitar o poliedro $\overline{\mathcal{F}}$. Um teorema de Weyls (veja [85]) inclusive demonstra a existência de um tal conjunto finito Π, tal que:

$$conv(\mathcal{F}) = \{x \in \mathbb{R}^n \mid \pi^T.x \leq \pi_0, \ \forall \ (\pi^T, \pi_0) \in \overline{\Pi}\}\,. \tag{4-4}$$

Assim, o problema de programação linear

$$\min\{f(x) = c.x \mid x \in \overline{\mathcal{F}},\ \pi^T.x \leq \pi_0\ \forall\ (\pi^T, \pi_0) \in \Pi\} \qquad (4\text{-}5)$$

é uma relaxação do problema P.

O procedimento básico para resolver este problema é começar com um subconjunto $\Pi' \subset \overline{\Pi}$ de desigualdades, que pode até ser vazio, e a cada iteração acrescentar novas desigualdades válidas ao mesmo. Se $\overline{x}$ é a solução ótima de 4-5, considerando-se apenas este subconjunto Pi', um segundo componente, chamado de algoritmo de separação, é utilizado para encontrar outro subconjunto $\Pi'' \subset \overline{\Pi}$ tal que todo par $(\pi, \pi_0) \in \Pi''$ viole a solução $\overline{x}$. Se Π'' é vazio, $\overline{x}$ é a solução ótima. Caso contrário, Π'' é adicionado a Π' e a resolução prossegue com mais uma iteração.

Se o elemento $\overline{x} \in \overline{\mathcal{F}}$ é a solução ótima da relaxação linear do problema P, e supondo-se que tal solução não satisfaça todas as restrições de integralidade, o problema da separação equivale a encontrar um plano de corte (um elemento $(\pi^T, \pi_0) \in \overline{\Pi}$) que não é satisfeito pela solução $\overline{x}$. A complexidade computacional deste problema varia bastante de acordo com a definição de $\overline{\Pi}$ e pode até ser equivalente à complexidade de resolver o problema original [60]. Assim, muitas vezes, utilizam-se heurísticas para separar tais desigualdades violadas.

Além dos parâmetros implícitos na descrição genérica do *branch-and-bound* da seção 4.1, os algoritmos *branch-and-cut* apresentam outros parâmetros, cujos ajustes também não costuma ser uma tarefa trivial. Dentre estes estão a determinação de quando e quantos cortes violados adicionar à formulação corrente e, da mesma forma, quando e quantos cortes excluir. Pode-se optar, por exemplo, por limitar o número de cortes gerados ou gerar cortes apenas para o problema original (conjunto $\overline{\mathcal{F}}$ de soluções), ou para os subproblemas gerados até a k-ésima iteração do método, ou para os k primeiros subproblemas, ou, ainda, de k em k escolhas de subproblemas. Observe que somente as desigualdades geradas para o subproblema da raiz são necessariamente válidas globalmente e, portanto, podendo ser utilizadas no processamento dos demais subproblemas. Desigualdades geradas para outros subproblemas podem ou não ser válidas [87] para os demais subproblemas.

A geração de cortes pode ser um processo que demanda um bom tempo de processamento, mesmo quando ao final não se consegue separar um corte violado. Além disso, a contínua adição de cortes aumenta o tamanho da formulação e, conseqüentemente, aumenta o tempo gasto na resolução de

sua relaxação linear. Assim, a sua utilização efetivamente só faz sentido quando ela é capaz de reduzir significativamente o número de subproblemas resolvidos.

4.3 *Branch-and-Price*

Analogamente ao descrito na seção 4.2, o algoritmo *branch-and-bound* (veja seção 4.1) também pode ser adaptado para, dado um problema de programação inteira (definido por 2-2), resolver relaxações lineares deste problema com um elevado número (potencialmente exponencial) de variáveis. Este refinamento é conhecido como *branch-and-price*.

Considere o problema de programação linear inteira IP, como formulado na seção 3.2. Aplicando-se a IP a reformulação tradicional para problemas de programação linear inteira, descrita na seção 3.2, obtém-se o chamado problema mestre IP'. O problema DWM (*Dantzig-Wolfe Master*), definido na seção 3.1, é obtido relaxando-se as restrições de integralidade em IP'. Um algoritmo *branch-and-price* para resolução do problema IP trabalha em cada nó da árvore de enumeração com uma versão restrita $DWMR$ de DWM, em que apenas um subconjunto de colunas é mantido na formulação corrente, conforme o esquema de geração de colunas descrito na seção 3.1.1.

Relembre-se aqui a necessidade de resolver até a otimalidade o DWM para cada subproblema criado, caso contrário a estimativa obtida pode não ser válida. Alternativamente, pode-se utilizar a estimativa lagrangeana [109] que produz uma estimativa inferior válida cada vez que o subproblema de geração de colunas é resolvido até a otimalidade.

A grande dificuldade, talvez a maior, no desenvolvimento de algoritmos *branch-and-price* é o estabelecimento de critérios adequados de *branching*. Um tal critério deve ser aplicável não somente ao problema mestre, mas também ao subproblema de geração de colunas. Em consequência, a verificação da existência de colunas de custo reduzido negativo deve satisfazer todos os critérios de *branching* utilizados no ramo atual da árvore de enumeração, desde o nó raiz até o nó corrente. Assim, o critério utilizado pode alterar a estrutura do subproblema usado na geração de colunas e tornar essa operação cada vez mais complicada.

Por exemplo, seja λ^* uma solução do problema DWM, em um certo nó da árvore de enumeração, que não satisfaz todas as restrições de integralidade. Os esquemas de *branching* mais utilizados em algoritmos

branch-and-bound e *branch-and-cut*, baseados na fixação do valor de uma variável em um valor inteiro, não pode ser aplicados a λ^* sem alterar a estrutura do subproblema de geração de colunas.

Diversos trabalhos ([16, 62, 97, 107, 108, 112]) exploram diferentes alternativas de *branching* em algoritmos *branch-and-price*, muitas delas específicas para problemas particulares.

4.4 *Branch-and-Cut-and-Price*

O desenvolvimento de algoritmos enumerativos associados a métodos de geração de cortes (algoritmos *branch-and-cut* – seção 4.2) e de geração de colunas (algoritmos *branch-and-price* – seção 4.3) é um campo razoavelmente bem explorado. Contudo, desde que a geração de cortes e a de colunas foram estabelecidas como duas das técnicas mais importantes na programação inteira, tem-se procurado maneiras de combiná-las de forma eficiente em um mesmo algoritmo.

O algoritmo *branch-and-cut-and-price* é uma especialização do *branch-and-bound* (veja seção 4.1) em que novas colunas e novas desigualdades válidas são geradas dinamicamemte à medida que a árvore de busca é percorrida. Embora este algoritmo utilize várias das técnicas usadas nos algoritmos *branch-and-cut* e *branch-and-price* (que essencialmente tem o mesmo princípio básico), o resultado dessa combinação requer técnicas muito mais sofisticadas do que as utilizadas em cada um em separado. Conforme descrito na seção 3.2, uma das razões é a necessidade de se acrescentar novas desigualdades (cortes) sem alterar a estrutura do subproblema de geração de colunas.

Em [94] Poggi de Aragão e Uchoa denominam *branch-and-cut-and-price* "robusto"a tais algoritmos em que a geração de novos nós na árvore de enumeração ou a adição de novas desigualdades não muda a estrutura dos subproblemas usados na geração de colunas. Dado que uma operação de *branching* pode ser vista como uma inserção de cortes nos vários subproblemas da árvore de enumeração, a mesma técnica usada na adição de cortes pode ser usada para *branching*.

Os bons resultados dos experimentos computacionais obtidos com a aplicação de algoritmos *branch-and-cut-and-price* robustos a problemas como *Cap-MST* (*Capacitated Minimum Spanning Tree*) [48], *CVRP* (*Capacitated Vehicle Routing*) [49] e *GAP* (*Generalized Assignment Problem*) [93] mostram o poder da combinação da geração de cortes e colunas.