MÉTODO DE ENUMERAÇÃO IMPLÍCITA EMPREGADO NA RESOLUÇÃO DE PROBLEMAS DE DECOMPOSIÇÃO

Shane Aparecida Soares Goulart

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Nelson Maculan Filho, D.Habil.

Profa. Maria Helena Jardim, D.Sc.

Prof. Luiz Satoru Ochi, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2001

GOULART, SHANE APARECIDA SOARES

Método de Enumeração Implícita Empregado na Resolução de Problemas de Decomposição [Rio de Janeiro] 2001

X, 45 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2001)

Tese — Universidade Federal do Rio de Janeiro, COPPE

1 - Método de Enumeração Implícita

2 - Problemas de Decomposição

I. COPPE/UFRJ II. Título (série)

Aos meus pais Vorlando e Geralda.

À minha irmã Alessandra.

Ao Anderson.

Agradecimentos

A Deus por tudo.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, pelo suporte financeiro recebido.

À Coordenação dos Programas de Pós-Graduação em Engenharia - COPPE/UFRJ, pela acolhida em seu ambiente de pós-graduação, que me proporcionou participar dos trabalhos de uma nova área de pesquisa.

Ao Prof. Nelson Maculan Filho, meu orientador, pela indicação do tema desta Tese, pelo interesse e pela ajuda prestada durante todo o desenvolvimento deste trabalho.

Aos Profs. Maria Helena Jardim e Luiz Satoru Ochi, por assentirem em compor a banca examinadora.

À Maria de Fátima Cruz Marques, pela atenção e dedicação na fase final deste trabalho.

Aos meus professores na COPPE/UFRJ, pelos conhecimentos transmitidos durante a minha passagem pelo programa de pós-graduação.

A todas as secretárias do Programa de Engenharia de Sistemas e Computação.

Ao amigo Elder Magalhães Macambira, por sua cortesia habitual e por várias informações úteis e impressindíveis que me ajudaram a concluir esta Tese.

Aos amigos da COPPE/Sistemas e de outros Programas, em especial à amiga Rosângela Dornas Torres, pela troca de informações e principalmente pelas dicas "quentes".

À bancada cearense, que se mostrou sempre disposta a me ajudar, especialmente Henrique Limaverde Cabral de Lima, Jorge Bergson Carvalho da Silva e Tibérius de Oliveira e Bonates.

Aos amigos Alexandre Soares Alves, Ana Mirtes Maciel Fouro, Moisés Teles Carvalho Júnior e Selma Foligne Crespio de Pinho, pelo companheirismo e pelos momentos alegres que passamos juntos.

Aos amigos Denise Candal dos Reis e Jurandir de Oliveira Lopes, pela ajuda fundamental no início do curso.

Ao colega Douglas Valiati, pela idéia inicial acerca do caminho pelo qual deveria seguir.

À Maria de Lourdes da Conceição, pelo carinho e pelos cafezinhos que me mantiveram acordada nessa longa jornada.

À D. Gercina Armando da Silva, por estar sempre pronta a ajudar com todo amor e carinho.

Aos meus pais, Vorlando Goulart e Geralda das Graças Soares Goulart, que me proporcionaram educação e estudo necessários para chegar onde estou.

À minha irmã Alessandra Soares Goulart Batista, que soube superar a distância e principalmente pela ajuda financeira que me trouxe para o Rio de Janeiro.

Ao Anderson Caldas Cailleaux, pelo incentivo e principalmente pelo ombro amigo onde pude chorar durante os momentos mais difíceis do desenvolvimento deste trabalho.

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

MÉTODO DE ENUMERAÇÃO IMPLÍCITA EMPREGADO NA RESOLUÇÃO DE PROBLEMAS DE DECOMPOSIÇÃO

Shane Aparecida Soares Goulart

Setembro/2001

Orientador: Nelson Maculan Filho

Programa: Engenharia de Sistemas e Computação

Neste trabalho, consideramos um problema mestre de Benders, onde a grande dificuldade reside no fato de que todas as soluções são viáveis. Caso fôssemos calcular todas as soluções, teríamos uma nova situação, que é a inviabilidade de tempo, principalmente para problemas com um número elevado de variáveis. Desta forma, aplicamos o método de enumeração implícita, o qual obtém a solução ótima sem que todas elas tenham que ser calculadas. Para isto, devemos associar uma variável a um nó, onde uma alternativa é utilizar alguns critérios para tal associação, chamados de heurísticas. Com base nos resultados fornecidos por estas heurísticas, considerando inclusive o número de nós avaliados e o tempo de cálculo até se obter a solução ótima, podemos ter conhecimento sobre qual delas teve um melhor desempenho em uma determinada situação que lhe foi apresentada.

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

IMPLICIT ENUMERATION METHOD USED IN
RESOLUTION OF DECOMPOSITION PROBLEMS

Shane Aparecida Soares Goulart

September/2001

Advisor: Nelson Maculan Filho

Department: Systems Engineering and Computer Science

In this work, we considered a Benders' master problem, where the great difficulty is the fact of all solutions are feasible. If we wanted to compute all the solutions, we'd have a new situation, that is the infeasibility of time, mainly to problems with a high number of variables. Then, we applied the implicit enumeration method, that gets the best solution without the computation of all of them. For this, we must associate a variable to a node, where a alternative is to use some criterion for this association, called heuristics. Based on the results given by these heuristics, considering also the number of avaliated nodes and the computation time until getting the best solution, we can have knowledge about which of them had the best performance in a specific situation that was presented.

Índice

Lista de Tabelas

Capítulo 1

Introdução

1.1 Apresentação

Considere o problema abaixo, denominado minmax:

$$(P) \text{ minimizar } \underset{i=1,2,\ldots,p}{\text{máximo}} \left\{ \alpha_i + \sum_{j=1}^{n} \beta_{ij} x_j \| x_j \in \{0,1\}, j = 1, \ldots, n \right\}.$$

onde α_i e β_{ij} são constantes $\in \Re$.

Aplicando o método de decomposição de Benders [Be62] no problema (P), podemos reescrever o problema mestre, o qual assumirá a forma abaixo:

$$\begin{array}{lll} (P) \quad \min & z & \\ & z \geq \alpha_i + \sum_{j=1}^{n} \beta_{ij} x_j & i = 1, 2, \ldots, p \\ & x_j \in \{0,1\} & j = 1, 2, \ldots, n \\ & z \in \Re, & \end{array}$$

onde α_i e β_{ij} são constantes $\in \Re$.

Pode-se perceber que quaisquer que sejam os valores das variáveis x_j fornecerão uma solução viável de (P), com $x_j \in \{0,1\}$ para $j = 1, 2, \ldots, n$. Por exemplo, se $n = 3$, tem-se que as soluções (0,0,0), (0,0,1), (0,1,0), (1,0,0), (0,1,1), (1,0,1), (1,1,0) e (1,1,1) são viáveis para o problema. Esta situação leva a um número de

2^n soluções viáveis.

1.2 Objetivo

Na Seção anterior, observamos um problema onde todas as soluções em x_j são viáveis, acarretando assim um grande número de soluções. Caso utilizemos um algoritmo que compare o valor de todas estas soluções para então fornecer a solução ótima do problema, e supondo que o cálculo de uma solução demore 0,00001 segundos, podemos concluir que, se este algoritmo for aplicado a um problema com 50 variáveis, ele demorará aproximadamente 350 anos para encontrar a solução ótima. Ou seja, a exploração de todas as soluções viáveis traz a inviabilidade de tempo. Logo, o objetivo deste trabalho é a busca de métodos que explorem um número pequeno de soluções e forneçam uma solução ótima para o problema, sendo portanto de extrema importância, principalmente em relação ao tempo de processamento para encontrar a solução ótima.

Para isto, será aplicado um método de enumeração implícita, onde o adotado neste trabalho foi o método *Branch and Bound*, sendo uma opção para que não sejam exploradas todas as combinações possíveis das variáveis e indicando que uma parte da árvore de busca não precisa ser explorada.

1.3 Organização da Tese

Esta tese está organizada do modo seguinte:

O Capítulo 2 apresenta a demonstração de como um problema de programação linear inteira mista possui um problema equivalente de programação linear inteira.

O Capítulo 3 apresenta o Método de *Branch and Bound* para Resolução do Problema de Benders, incluindo algumas heurísticas que serão utilizadas para a resolução deste problema.

O Capítulo 4 apresenta os resultados computacionais obtidos, de acordo com os critérios introduzidos para cada instância.

O Capítulo 5 apresenta a conclusão desta Tese.

Capítulo 2

Método de Decomposição de Benders

2.1 Introdução

Usando a teoria da dualidade em programação linear, é possível mostrar que qualquer problema inteiro misto pode ser escrito como um problema inteiro. Isto sugere que seja resolvido um problema inteiro misto através da resolução de seu problema inteiro equivalente. Entretanto, é difícil obter computacionalmente todas as restrições do problema inteiro, em caso de haver a necessidade do valor numérico de um ponto extremo ou de um raio extremo [Ga94, NeWo88] em um poliedro convexo [Ga94, NeWo88], além de existirem tantas restrições quanto pontos extremos ou raios extremos. Como o poliedro convexo pode ter um número elevado de pontos extremos, não podem ser listadas todas as restrições. Para superar esta dificuldade, J. F. Benders [Be62] propôs uma técnica na qual o problema inteiro equivalente é resolvido após gerar apenas um subconjunto de restrições. Isto é, as restrições que não foram "enumeradas implicitamente" não representam nenhuma restrição ao problema inteiro. O algoritmo de particionamento de Benders [Be62] trabalha através de resoluções sucessivas de um problema linear e de um problema inteiro. O problema linear produz um ponto extremo ou um raio extremo e uma única restrição para o problema inteiro. Também, o valor de sua solução ótima dá

um limite superior para a solução ótima para o problema inteiro misto. Quando resolvido, o problema inteiro, que é análogo ao problema inteiro misto quando tiver todas as suas restrições, produz um limite inferior não-decrescente. Quando os dois limites coincidem, a solução ótima inteira mista foi encontrada e o processo termina.

2.2 Método de Decomposição de Benders em Programação Linear Inteira Mista

Considere o problema de programação linear inteira mista:

$$
\begin{aligned}
(P_1): \quad & \text{minimizar} && cx + dy && (1)\\
& \text{sujeito a:} && Ax + Dy \geq b && (2)\\
& && x \in Z_+^n && (3)\\
& && y \geq 0, && (4)
\end{aligned}
$$

onde:

- $c = (c_j)$ é um vetor de n linhas;
- $d = (d_j)$ é um vetor de r linhas;
- $A = (a_{ij})$ é uma matriz m x n;
- $D = (d_{ij})$ é uma matriz m x r;
- $b = (b_i)$ é um vetor de constantes com m colunas;
- $x = (x_j)$ é um vetor de n variáveis inteiras;
- $y = (y_i)$ é um vetor de r variáveis contínuas;
- Z_+^n é o conjunto de vetores inteiros não-negativos de dimensão n.

Em muitos casos, as variáveis inteiras x_j são usadas para representar relacionamentos lógicos, o que faz com que sejam limitadas, por convenção, aos valores 0

ou 1. Nestes casos, as restrições (3) devem ser substituídas por $x \in B^n$, onde B^n é o conjunto de vetores binários (0 ou 1) de dimensão n. Desta forma, o problema (P_1) pode ser escrito como:

$$
\begin{array}{lll}
(P_2): & \text{minimizar} & cx + dy \quad (5) \\
& \text{sujeito a:} & Ax + Dy \geq b \quad (6) \\
& & x \in \{0,1\}^n \quad (7) \\
& & y \geq 0, \quad (8)
\end{array}
$$

Fixando $x = \overline{x} \in \{0,1\}^n$. Assim, (P_2) será reescrito da forma seguinte:

$$
\begin{array}{lll}
(P_3): & \text{minimizar} & c\overline{x} + dy \quad (9) \\
& \text{sujeito a:} & Dy \geq b - A\overline{x} \quad (10) \\
& & y \geq 0. \quad (11)
\end{array}
$$

Se existir um y igual a $\overline{y}$ que satisfaça às restrições (10) e (11), pode-se afirmar que o problema (P_3) admitirá $\overline{y}$ como solução. Assim, $(\overline{x},\overline{y})$ será uma solução viável do problema (P_2).

Aplicando o lema de Farkas e Minkowski [Mi86], que associa uma variável dual v_i (sem imposição de sinal) a cada uma das restrições i de (P_3), e sendo v um vetor linha de variáveis duais com dimensão igual ao número de restrições, pode-se afirmar que: (P_3) tem uma solução $y \geq 0$ se e somente se $v(b - Ax) \leq 0$ para todo v que assegure que $vD \leq 0$.

Em vez de trabalharmos com restrições que dependem de $\overline{x}$, tiraremos o dual de (P_3).

Segue-se que o dual do problema (P_3) é:

$$(D_1): \quad \text{maximizar} \quad c\overline{x} + u(b - A\overline{x}) \tag{12}$$

$$\text{sujeito a:} \quad uD \leq d \tag{13}$$

$$u \geq 0, \tag{14}$$

onde:
- $u^T \in \Re^m$ é o vetor linha do dual;
- $c\overline{x}$ é uma constante.

As restrições (13) e (14) independem de $\overline{x}$, formam um conjunto fechado e limitado e não têm raios extremos (apenas pontos extremos ou vértices do poliedro).

Seja um cone $C = \{v \mid vD \leq 0\}$ poliédrico e que tenha um número finito de geradores, definidos por $v^1, \ldots, v^S$, onde todos $v \in C$ são uma combinação finita com coeficientes não-negativos de v^s para $s = 1, \ldots, S$. Desta forma, pode-se dizer que o lema de Farkas e Minkowski [Mi86] é equivalente ao sistema de desigualdades:

$$\begin{aligned} v^1(b - Ax) &\leq 0 \\ v^2(b - Ax) &\leq 0 \\ &\cdots \\ v^S(b - Ax) &\leq 0 \end{aligned} \tag{15}$$

Caso o sistema (15) não tenha nenhuma solução em x, isto significa que não existe solução para (P_3). Porém, assumiremos que esta solução exista.

Chamemos o conjunto de restrições de (D_1) de $U = \{u \geq 0 \mid uD \leq d\}$, onde podemos observar que U não depende de x e que os raios extremos são $(v^1, \ldots, v^S)$.

Definam-se:

(i) u^p para $p = 1, 2, \ldots, P$ os vértices de U;

(ii) v^s para $s = 1, 2, \ldots, S$ os raios extremos de U.

Com isso, podemos escrever $\forall u \in U$:

$$u = \sum_{p=1}^{P} \lambda_p u^p + \sum_{s=1}^{S} \mu_s v^s, \quad \text{onde} \quad \sum_{p=1}^{P} \lambda_p = 1, \; \lambda_p \geq 0 \quad \text{para } p = 1, 2, \ldots, P$$

$$\text{e} \qquad \mu_s \geq 0 \qquad \text{para } s = 1, 2, \ldots, S.$$

Segundo a teoria da dualidade em programação linear, pode-se constatar que:

1. Se $U = \varnothing$, isto é, não admitir solução viável, (P_3) e (D_1) serão ilimitados ou não admitirão soluções viáveis. Porém, foi assumido que (P_3) tenha uma solução. Assim, se $U = \varnothing$, isto é por que (P_3) é ilimitado para todos os valores de x;

2. Se (D_1) admitir solução ótima, ao menos uma será um vértice de U;

3. Se (D_1) não possuir solução ótima limitada para $x = \overline{x}$, isto é, existe um raio vetor v^s tal que $v^s(b - Ax) > 0$. Neste caso, (P_3) será vazio e não existirá $(\overline{x}, y)$ viável para (P_2). Isto fará com que nos interessemos pelas soluções $x \in \{0,1\}^n$ tais que $v^s(b - Ax) \leq 0$, para $s = 1, 2, \ldots, S$. Pois, caso estes x sejam também não-negativos e inteiros, serão também soluções viáveis de (P_2), podendo ser ilustrado da seguinte maneira: consideremos a função objetivo de (D_1), isto é, $t = c\overline{x} + u(b - A\overline{x})$. Se v^s é um raio extremo de U para o qual $v^s(b - A\overline{x}) > 0$, então seja $\widehat{u} = \overline{u} + \lambda v^s$ para $\lambda > 0$ onde $\overline{u} \in D_1$. Sabemos que $\widehat{u} \in U$ e façamos $u = \widehat{u}$ na função objetivo de (D_1): $t = c\overline{x} + \widehat{u}(b - A\overline{x})$. Ou ainda: $t = c\overline{x} + (\overline{u} + \lambda v^s)(b - A\overline{x}) = c\overline{x} + \overline{u}(b - A\overline{x}) + \lambda v^s(b - A\overline{x})$. Se $\lambda \to \infty$, então $t \to \infty$, pois $v^s(b - A\overline{x}) > 0$.

Supondo $U \neq \varnothing$ e sendo $u^1, \ldots, u^P$ os vértices de U, o problema (D_1) poderá ser transformado em um outro problema:

$$(D_2): \quad \text{maximizar} \quad u^p(b - Ax), \quad p = 1, 2, \ldots, P \tag{16}$$

$$\text{sujeito a:} \quad v^s(b - Ax) \leq 0, \quad s = 1, 2, \ldots, S \tag{17}$$

$$x \in \{0,1\}^n. \tag{18}$$

Apenas os vetores x que satisfaçam às restrições (17) e (18) darão (x,y) viável de (P_2). Desta forma, pode-se escrever (P_2) do modo seguinte:

$$(P)\text{: } \underset{x\in\{0,1\}^n}{\text{minimizar}} \left[cx + \underset{p=1,2,\ldots,P}{\text{máximo}} \{u^p(b-Ax) \mid v^s(b-Ax) \leq 0,\ s=1,2,\ldots,S\} \right]$$

$$\text{Seja } z = \left[cx + \underset{p=1,2,\ldots,P}{\text{máximo}} \{u^p(b-Ax) \mid v^s(b-Ax) \leq 0,\ s=1,2,\ldots,S\} \right].$$

$$\begin{array}{lllll} \text{Então,} & z \geq cx & + & u^p(b-Ax) & p=1,2,\ldots,P \\ & & & v^s(b-Ax) & s=1,2,\ldots,S. \end{array}$$

Portanto, reescreve-se (P_3) como sendo

$$\begin{array}{lllll} (P_4)\text{:} & \text{minimizar} & z & & (19) \\ & \text{sujeito a:} & z \geq cx + u^p(b-Ax) & p=1,2,\ldots,P & (20) \\ & & x \in \{0,1\}^n. & & (21) \end{array}$$

Acrescentando o sistema (15) às restrições de (P_4), pode-se escrever este problema da forma seguinte:

$$\begin{array}{lllll} (P_5)\text{:} & \text{minimizar} & z & & (22) \\ & \text{sujeito a:} & z \geq cx + u^p(b-Ax) & p=1,2,\ldots,P & (23) \\ & & v^s(b-Ax) \leq 0 & s=1,2,\ldots,S & (24) \\ & & x \in \{0,1\}^n. & & (25) \end{array}$$

Pode-se verificar que (P_5) é um problema de programação linear $0-1$, a menos de z, que é contínua.

Portanto, deve-se resolver este problema (obter uma solução x^*) e levá-la a

(P_3), obtendo y^*. Este par de soluções (x^*,y^*) é a solução de (P_2).

2.3 Algoritmo de Decomposição de Benders

O principal objetivo para a concepção de técnicas de decomposição matemática é conseguir resolver problemas muito difíceis ou muito grandes, através da solução repetida de uma série de problemas mais fáceis ou menores. Um exemplo é a decomposição de Benders [Be62], na qual um problema de programação linear inteiro misto é decomposto em uma série de subproblemas menores, um denominado de *Mestre* (um problema de programação linear inteiro misto) e outro chamado de *Escravo* (um problema de programação linear).

A teoria de decomposição de Benders é um tópico que já foi bastante estudado pela comunidade científica. Vários são os artigos, tutoriais e capítulos de livros sobre este assunto. Entre diversas referências, podemos citar, por exemplo, [BaJaSh90, Gr75, Ma77, Mi86, NeWo88, SaKa89, Ta75].

É interessante observar o algoritmo de Decomposição de Benders [Be62], mostrado a seguir em suas 5 fases, onde a idéia é simplificar todo o processo que "transformou" (P_2) em (P_5).

Fase 1:

Em (P_2), adotar um vetor que satisfaça à restrição (7). Definir z^{sup} como $+\infty$ e z^{inf} como $-\infty$. Continuar a execução do algoritmo.

Fase 2:

Resolver o problema (D_1) utilizando os valores adotados na Fase 1. Duas situações podem ocorrer: (1) se (D_1) não admitir solução viável, pare a execução do algoritmo, pois (P_2) será ilimitado ou vazio; (2) em caso contrário, o problema (P_2) terá solução limitada ou ilimitada.

Fase 2a:

Se a solução for limitada, (P_2) fornece valores para os vértices u^p, onde z^{sup} deve assumir o valor da equação $c\overline{x} + u^p(b - A\overline{x})$, como definido em (P_2), caso o valor anterior desta equação fosse maior do que este recém calculado. Continuar a execução do algoritmo a partir da Fase 3a.

Fase 2b:

Se a solução for ilimitada, (P_2) fornece valores para os vértices u^p e para os raios extremos v^s. Continuar a execução do algoritmo a partir da Fase 3b.

Fase 3:

Esta fase está subdividida em:

Fase 3a:

Acrescentar ao problema (P_5) uma restrição do tipo $z \geq cx + u^p(b - Ax)$.

Fase 3b:

Acrescentar ao problema (P_5) uma restrição do tipo $z \geq cx + u^p(b - Ax)$ e uma outra do tipo $v^s(b - Ax) \leq 0$.

Após estas três fases, teremos um problema como (P_5). Pode ser destacado que, se continuássemos com este algoritmo, poderíamos ter um número maior de restrições acrescentadas ao problema final, a medida que não fosse sendo encontrado o par $(\overline{x},\overline{y})$ que fornecesse a solução ótima para (P_2). Porém, o objetivo desta tese não é encontrar a solução ótima através do algoritmo de Decomposição de Benders [Be62, Ma77, PaRa88, Zi74], mas sim através do Método de *Branch and Bound* [GoLu2000, Ma77, Sa75, Ta75, Wo98]. Portanto, após ter partido de um problema como (P_1) ou como (P_2), foi demonstrado como este problema pode alcançar a forma de um problema (P_5).

2.4 Problema de Benders

Comparando o problema (P_2) com o problema (P_5), é possível perceber que este último já está em função apenas da variável inteira x, ou seja, foi demonstrado como um problema de programação linear inteira mista possui um problema equivalente de programação linear inteira.

Com estes procedimentos iniciais, podemos fazer (i) $u^p(b-Ax)$ e (ii) $v^s(b-Ax)$. Mas nesta tese, apenas (i) será considerada para resolução. Além disso, cada elemento desta multiplicação pode ser dimensionado da forma seguinte:

- u^p é um vetor linha com m colunas;
- $(b - Ax)$ é um vetor coluna com m linhas.

Desta forma e após ser efetuada a multiplicação, tem-se que (i) pode ser resumido através da adição de uma constante α_i com o somatório do produto de uma constante β_{ij} pela variável inteira x_j, onde $i = 1, 2, \ldots, p$ é o número de restrições e $j = 1, 2, \ldots, n$ é o número de variáveis inteiras.

Logo, o problema (P_5) pode ser escrito como:

$$
\begin{aligned}
(P_6): \quad \text{minimizar} \quad & z && && (26)\\
& z \geq \alpha_i + \sum_{j=1}^{n} \beta_{ij} x_j && i = 1, 2, \ldots, p && (27)\\
& x_j \in \{0, 1\} && j = 1, 2, \ldots, n && (28)\\
& z \in \Re, && && (29)
\end{aligned}
$$

onde α_i e β_{ij} são constantes pertencentes a $\Re$.

O próximo passo, a ser visto no Capítulo 3, será resolver o problema (P_6) através do Método de *Branch and Bound* [GoLu2000, Ma77, Sa75, Ta75, Wo98], isto é, encontrar valores para as variáveis x_j para $j = 1, 2, \ldots, n$ e que sejam a solução ótima deste problema. E após encontrar estes valores, deve-se retornar

ao problema (P_3) para obter y. Com todos os valores das variáveis inteiras e contínuas já calculados, podem-se aplicá-los na função objetivo (9) e chegar ao melhor resultado que o problema pode fornecer.

Capítulo 3

Método de *Branch and Bound* para Resolução do Problema de Benders

3.1 Problema de Benders

Conforme visto no Capítulo 2, o problema

$$\begin{aligned} (P) \quad & \text{minimizar} && cx + dy \\ & \text{sujeito a:} && Ax + Dy \geq b \\ & && x \in Z_+^n \\ & && y \geq 0 \end{aligned}$$

pode também ser escrito de acordo com o modelo matemático seguinte:

$$\begin{aligned} (P) \quad & \text{minimizar} && z \\ & && z \geq \alpha_i + \sum_{j=1}^{n} \beta_{ij} x_j && i = 1, 2, \ldots, p \\ & && x_j \in \{0, 1\} && j = 1, 2, \ldots, n \\ & && z \in \Re, \end{aligned}$$

onde α_i e β_{ij} são constantes pertencentes a $\Re$.

Com base neste último modelo matemático, serão abordadas as características do problema (P) na Seção [3.1.1] e um método de enumeração que resolva este problema na Seção [3.1.2].

3.1.1 Características

Podem ser mencionadas três características fundamentais deste problema: **viabilidade, soluções descendentes** e **dependência das variáveis**, a serem definidas a seguir.

3.1.1.1 Viabilidade

Esta característica afirma que todas as soluções são viáveis, isto é, a variável x_j para $j = 1, 2, \ldots, n$ pode assumir qualquer valor em seu domínio (0 ou 1) que, mesmo assim, fornecerá uma solução viável.

Como exemplo, pode-se imaginar um primeiro problema, no qual n seja igual a 3. Tendo em vista que o número de soluções viáveis é dado pela fórmula 2^n, tal fórmula proporcionaria 8 soluções viáveis, quais seriam: (0,0,0), (0,0,1), (0,1,0), (1,0,0), (0,1,1), (1,1,0), (1,0,1) e (1,1,1). Do mesmo modo, caso houvesse um segundo problema que apresentasse $n = 50$, as soluções viáveis seriam aproximadamente 1×10^{15}.

Seja um algoritmo que calcule o valor de todas as soluções e compare-as a seguir, com a finalidade de fornecer a solução ótima para o problema. Supondo que o cálculo de cada solução demorasse $0,00001$ segundo para ser realizado, ter-se-ia que a solução ótima seria encontrada após o intervalo de tempo de 350 anos. Portanto, pode-se concluir que a exploração de todas as soluções viáveis para um determinado problema pode trazer a inviabilidade de tempo, caso este problema apresente um número elevado de soluções viáveis, o que pode ser resolvido ao ser(em) encontrado(s) método(s) que não explore(m) todas as soluções e, apesar disso, forneça(m) a solução ótima para o problema.

Uma alternativa é um método de enumeração que, ao ser associado a uma árvore de busca, enumera as soluções até que seja encontrada a solução ótima.

Ao se utilizar um método de enumeração **implícita**, tem-se uma opção para que não sejam exploradas todas as combinações possíveis de variáveis, fazendo com que uma parte da árvore de busca não precise ser explorada. Porém, tendo em vista que uma das formas de execução do método de enumeração implícita é não calcular as soluções inviáveis, deve-se relembrar que este problema fornece todas as soluções como viáveis.

3.1.1.2 Soluções descendentes

As soluções descendentes representam uma característica que afirma que a solução ótima é independente do nível da árvore de busca onde ela se encontra.

Sejam:

1. v um determinado nó da árvore de busca;
2. T_1 a sub-árvore esquerda de v;
3. T_2 a sub-árvore direita de v.

Será adotado o critério seguinte para a árvore de busca: T_1 terá os valores das variáveis iguais àqueles de v, enquanto T_2 alterará tais valores, sendo uma alteração para cada nó.

Uma solução descendente de v consiste na solução de um nó u que seja pertencente a T_1 ou T_2. Entretanto, não se pode afirmar que as soluções descendentes deste nó u são melhores ou piores do que a melhor solução encontrada até o momento, mesmo que a solução em tal nó seja melhor ou pior.

Por exemplo:

$$
\begin{aligned}
(EX) \quad \min \quad & z \\
& z \geq 0 - 4x_1 + 5x_2 + 3x_3 \\
& z \geq 0 + 6x_1 - 3x_2 - 7x_3 \\
& x_j \in \{0,1\} \qquad\qquad j = 1,2,3 \\
& z \in \Re.
\end{aligned}
$$

Inicializando todas as variáveis com valor 0 e chamando este primeiro nó da árvore de busca de nó 1, tem-se que $z_1 = 0$.

Percorrendo o caminho de T_2 e alterando o valor de x_1 para 1, obtém-se o nó 2, o qual fornece $z_2 = 6$.

Comparando os dois valores calculados, tem-se que $z_1 < z_2$. Entretanto, esta comparação não indica que não haja uma solução melhor do que z_1, mesmo que seja descendente do nó 2. Como exemplo, pode-se desenvolver a seqüência seguinte: do nó 2, pode-se alcançar o nó 3 de acordo com T_1 ($x_1 = 1$, $x_2 = 0$ e $x_3 = 0$), o qual fornece $z_3 = 6$; do nó 3, pode-se alcançar o nó 4 de acordo com T_2 ($x_1 = 1$, $x_2 = 0$ e $x_3 = 1$), o qual fornece $z_4 = -1$.

Conforme visto, o nó 4 é descendente do nó 2 e fornece uma solução melhor do que o nó 1.

Neste exemplo, inicializaram-se todas variáveis com valor 0. Porém, pode acontecer a mesma situação se todas as variáveis forem inicializadas com valor 1, ou até mesmo com uma combinação de 0 e 1.

3.1.1.3 Dependência das variáveis

Esta característica afirma que as variáveis têm um alto grau de dependência, isto é, o valor de uma variável influencia o valor de outra, impossibilitando o aproveitamento de resultados anteriormente calculados.

Como exemplo, pode-se utilizar o problema (EX):

1. Fixando-se $x_3 = 0$, tem-se que a solução ótima acontece no nó onde as variáveis assumem os valores seguintes: $x_1 = 0$, $x_2 = 0$ e $x_3 = 0$. Estes valores fornecem $z = 0$.

2. Fixando-se $x_3 = 1$, tem-se que a solução ótima acontece no nó onde as variáveis assumem os valores seguintes: $x_1 = 1$, $x_2 = 0$ e $x_3 = 1$. Estes valores fornecem $z = -1$.

Portanto, ao ser alterado o valor de uma variável, não é suficiente que sejam comparadas apenas as soluções para saber qual delas é a solução ótima, pois, caso

este procedimento tivesse sido adotado, ter-se-ia que $z = 0$ para $x_3 = 0$ e $z = 3$ para $x_3 = 1$, o que poderia proporcionar a solução ótima quando $x_n = 0$ para $n = 1, 2, 3$.

Desta forma, chega-se à conclusão de que o processo de enumeração deve continuar assim que for alterado o valor de uma variável.

3.1.2 Método de Enumeração Implícita

Geralmente, pode-se considerar que o espaço formado pelas soluções de um problema inteiro possui um número finito de pontos viáveis possíveis. Um método direto para resolver problemas inteiros é enumerar *explicitamente* todos estes pontos. Neste caso, a solução ótima é determinada pelo(s) ponto(s) que produzir(em) o melhor valor da função objetivo: máximo ou mínimo.

O problema óbvio da técnica anterior é que o número de pontos da solução pode se tornar impraticavelmente muito grande, onde o resultado é que a solução talvez não seja determinada em um tempo razoável. A idéia da enumeração *implícita* está baseada em considerar apenas uma porção pequena de todos os pontos possíveis para solução, enquanto automaticamente afasta a necessidade de cálculo dos outros pontos.

Dois aspectos importantes para uma implementação bem sucedida da enumeração implícita devem ser destacados:

1. Existe a necessidade por uma enumeração que assegure que todos os pontos possíveis sejam enumerados (implicitamente) de um modo não-redundante;

2. O número de testes que "eliminem" algumas soluções deve ser projetado para excluir o número de soluções que for possível.

Conforme abordado na Seção [3.1.1.1], o problema (P) não oferece a possibilidade de não serem calculadas as soluções inviáveis, em razão de todas elas serem viáveis, dificultando assim a possibilidade de uma solução rápida.

O método fornece duas formas de enumeração implícita, que são realizadas através da fixação de variáveis e da utilização dos limites inferior e superior.

3.1.2.1 Fixação de variáveis

A enumeração implícita através da fixação de variáveis indica que a solução ótima está em apenas uma das sub-árvores descendentes de um nó v, podendo esta sub-árvore ser a esquerda ou a direita.

A enumeração implícita é obtida se a variável atender a uma das condições seguintes:

$$(1) \quad \exists j \mid \beta_{ij} \leq 0 \quad \forall i \quad \rightarrow \quad \text{então } x_j = 1$$

Como a solução ótima está na sub-árvore direita, esta condição indica que a sub-árvore esquerda do nó v associado a x_j não precisa ser explorada, fornecendo assim a enumeração implícita;

$$(2) \quad \exists j \mid \beta_{ij} \geq 0 \quad \forall i \quad \rightarrow \quad \text{então } x_j = 0$$

Como a solução ótima está na sub-árvore esquerda, esta condição indica que a sub-árvore direita do nó v associado a x_j não precisa ser explorada, fornecendo assim a enumeração implícita.

3.1.2.2 Limites inferior e superior

A enumeração implícita através da utilização de um limite superior indica que cada nó da árvore de busca fornece uma solução viável para o problema, eliminando a necessidade de serem empregados os limites superiores locais, os quais são valores maiores ou iguais ao maior valor que uma solução descendente de um determinado nó pode assumir.

Já os limites inferiores locais são valores menores ou iguais ao menor valor que uma solução descendente de um determinado nó pode assumir.

No caso da enumeração implícita, serão utilizados o limite superior do problema

e o limite inferior local, onde o primeiro é a melhor solução encontrada até o momento e será denominado $\hat{Z}$ e o segundo afirma que cada nó v terá um limite inferior a ele associado e será denominado Z'_v. Esta enumeração implícita é obtida se, em um determinado nó v, a condição seguinte for atendida:

$$Z'_v \geq \hat{Z}.$$

Esta condição indica que nenhuma das sub-árvores precisam ser exploradas, em razão de a solução de todos os nós descendentes serem maiores ou iguais à melhor solução encontrada até o momento, fazendo com que seja desnecessário explorar as sub-árvores.

Para que ocorra um número maior de enumeração implícita, é importante que se encontre um bom limite superior rapidamente. Para isto, pode-se fazer um processamento nas variáveis e nos valores, isto é, escolher a variável a ser instanciada e qual será o valor a ela fixado primeiramente (0 ou 1). Em se tratando da árvore de busca, é o equivalente a escolher qual será a variável associada ao nó em questão e qual será a sub-árvore a ser primeiramente explorada.

Cálculo do limite inferior. O cálculo do limite inferior é dado pela resolução do problema seguinte:

$$\begin{aligned} (LI) \quad \max \quad & z \\ & z = \alpha_i + \sum_{j \in N} \beta_{ij} + \sum_{j \in J} \beta_{ij} + (\sum \beta_{ij} \mid j \in C \text{ e } \beta_{ij} < 0) \\ & i = 1, 2, \ldots, p \\ & z \in \Re, \end{aligned}$$

onde:

- N é o conjunto das variáveis cujos coeficientes são todos negativos, isto é, conjunto das variáveis que atendem à condição (1) da Seção [3.1.2.1];
- P é o conjunto das variáveis cujos coeficientes são todos positivos, isto é, conjunto das variáveis que atendem à condição (2) da Seção [3.1.2.1];

- J é o conjunto das variáveis cujos valores são fixados em 1 na tentativa de se encontrar o ótimo;
- C é o conjunto das variáveis cujos valores são indefinidos.

3.2 Método *Branch and Bound*

Apresentaremos a seguir uma idéia geral do método *Branch and Bound.*

Um dos métodos mais usados em Programação Inteira é o método *Branch and Bound*, o qual é um método exato de otimização e que utiliza a programação linear como ferramenta para que se obtenha uma solução para o problema, sem considerar a restrição onde as variáveis devessem assumir apenas valores inteiros.

O objetivo deste método é resolver problemas como se fossem problemas de programação linear e, conforme mencionado no parágrafo anterior, sem considerar a restrição onde as variáveis devessem assumir apenas valores inteiros, começando a utilizar alguma rotina de Programação Linear (PL). Chamaremos esta primeira iteração de iteração inicial, onde, em caso de a solução ser inteira, teremos a solução para o problema, de modo que a solução obtida com o PL seja a solução ótima do problema inteiro. Em caso contrário, será necessário que se continue com a procura pela solução do problema inteiro.

Se, na iteração inicial, a solução obtida não for inteira, devemos proceder a seleção de uma variável não-inteira x_i, encontrando-se os dois inteiros mais próximos de x_i. Sejam estes inteiros x_{MENOR} e x_{MAIOR} os inteiros menor do que x_i e maior do que x_i, respectivamente. O método forma duas novas restrições ao problema original, de modo que sejam obtidos dois novos problemas. Este procedimento é chamado de *ramificação.* A seguir, procede-se a busca por uma solução inteira em cada um destes problemas, para os quais, como na iteração inicial, será usada a PL para resolver os problemas de programação linear correspondentes.

Para sua análise, o método *Branch and Bound* utiliza um limite inferior, que será aqui denominado de LI, examinando apenas aqueles problemas cuja solução proporcionada pela PL esteja acima do LI. Assim sendo, é necessário que o algo-

ritmo de *Branch and Bound* determine um LI o mais rapidamente possível. Uma forma prática é começar com um LI para os valores das variáveis x iguais ao seu limite inferior. Existem ainda rotinas de *Branch and Bound* que esperam obter um LI igual à primeira solução que tenha apenas valores inteiros. Observemos que, no caso da iteração inicial, o processo termina se a solução tiver apenas valores inteiros. Este método troca seu LI quando detecta que há um problema com uma solução inteira cujo valor da função objetivo seja maior do que o LI atual.

Uma vez determinado o LI, se não corresponder à iteração inicial, devemos obter novos problemas utilizando o processo de particionamento. Somente são analisados aqueles problemas que tenham como solução um valor para a função objetivo que seja maior do que o último LI determinado. O método *Branch and Bound* termina quando não existem mais problemas para analisar.

O método *Branch and Bound* baseia-se no princípio de que os problemas a serem resolvidos apresentam uma porcentagem de coeficientes positivos maior ou igual à porcentagem de coeficientes negativos. Se este princípio não se concretizar, devem ser feitas pequenas adaptações quando do cálculo do limite inferior, assim como quando da escolha da variável a ser instanciada.

3.2.1 Heurísticas para escolha de variáveis e valores

Para que se tenha uma solução em cada nó da árvore de busca e considerando-se os conjuntos como definidos, temos que dar valores às variáveis do conjunto C, que são indefinidas. Como o princípio previamente estabelecido é de que o problema tenha mais coeficientes positivos do que negativos e devido ao fato de que é mais provável que o valor da variável seja 0 na solução ótima, todas as variáveis pertencentes ao conjunto C são inicializadas com valor 0.

Este procedimento define que a sub-árvore direita de v será primeiramente explorada, uma vez que, ao ser explorada a sub-árvore esquerda, o valor de x associado a v não será alterado. Assim sendo, uma boa escolha de qual variável deverá ser associada a um determinado nó seria uma em que seu valor na solução ótima seja 1. Para isso, uma alternativa é associar uma variável a um nó utilizando

alguns critérios que possam indicar que tal variável tenha valor 1 na solução ótima. A estes critérios chamaremos de heurísticas.

Uma heurística é uma técnica que busca alcançar uma boa solução utilizando um esforço computacional considerado razoável, sendo capaz de garantir a viabilidade ou a otimalidade da solução encontrada ou, ainda, em muitos casos, ambas, especialmente nas ocasiões em que essa busca partir de uma solução viável próxima ao ótimo.

Para associarmos variáveis aos nós definimos 5 heurísticas, que estão descritas a seguir:

PrimeiraVariável:

Esta heurística associa a primeira variável do conjunto C ao nó v. Esta rotina tem complexidade $O(1)$.

MínimoMáximo:

Nesta heurística, o primeiro passo é encontrar a expressão máxima, considerando a soma do termo independente α com o somatório dos coeficientes das variáveis pertencentes a N e J. O segundo passo é encontrar, na expressão máxima, o menor coeficiente das variáveis pertencentes a C. A variável referente a este coeficiente será associada ao nó v. Esta rotina tem complexidade $O(np)$.

MenorSomaColuna:

Esta heurística soma todas as colunas referentes às variáveis pertencentes ao conjunto C, onde a variável referente à coluna que fornecer a menor soma será associada ao nó v. Esta rotina tem complexidade $O(np)$.

MaiorNegativoColuna:

Esta heurística associa ao nó v a variável pertencente a C cuja coluna tem mais números negativos. Esta rotina tem complexidade $O(np)$.

MáximoSomaLinha:

Nesta heurística, o primeiro passo é encontrar a expressão máxima, considerando a soma dos termos em C. O segundo passo é encontrar, na expressão máxima, o menor coeficiente das variáveis pertencentes a C. A variável referente a este coeficiente será associada ao nó v. Esta rotina tem complexidade $O(np)$.

Com base no exposto acima, pode-se concluir que a primeira heurística não apresenta nenhum critério para que seja associada uma variável a um nó, enquanto as outras executam um processamento nas informações, para que então seja feita a associação que possa fornecer um bom resultado.

3.2.2 Condições de Parada e *Backtracking*

Condições de Parada:

Durante a exploração da árvore de busca, há duas formas de se efetuar a parada: quando se chega a uma folha (isto é, assim que o conjunto C ficar vazio) e quando ocorre a enumeração implícita (isto é, assim que não for necessária a exploração de uma ou das duas sub-árvores de um nó v).

***Backtracking*:**

O *backtracking* em um nó v da árvore de busca é o retorno para o pai de v, indicando assim que todas as soluções descendentes deste nó v foram exploradas. Ocorre *backtracking* em um nó v da árvore de busca, se não houver mais sub-árvores a serem exploradas, sendo v a raiz.

Então, a condição de parada do algoritmo é quando ocorrer o *backtracking* no nó raiz da árvore de busca, isto é, quando as duas sub-árvores do nó raiz da árvore de busca já tiverem sido exploradas.

3.2.3 Algoritmo de Enumeração

Será utilizada a estrutura de pilha proposta por Glover e Geoffrion, respectivamente em 1965 e 1967, onde a pilha representa o conjunto de índices associados às variáveis fixadas.

Seja a pilha π, para a qual $p(j)$ seja sua j-ésima componente, tal que:

$$\begin{aligned} p(j) > 0 \quad &\text{se} \quad x_{p(j)} = 1 \\ p(j) < 0 \quad &\text{se} \quad x_{p(j)} = 0. \end{aligned}$$

Esta pilha, juntamente com os conjuntos anteriormente definidos representarão a busca na árvore; os conjuntos N e P representarão valores fixos, os quais permanecerão do início do algoritmo até seu final; já os conjuntos C e J mudam conforme a busca for procedendo. O conjunto C é inicializado com todas as variáveis, exceto com aquelas pertencentes aos conjuntos N e P, enquanto o conjunto J é inicializado como vazio.

As modificações nos conjuntos C e J procedem da forma seguinte:

1. Quando for inserida uma variável na pilha, retira-se esta de C e insere-se em J;

2. Quando for alterado o valor de um elemento da pilha para negativo, retira-se a variável associada ao elemento do conjunto J;

3. Quando for retirado um elemento da pilha, a variável a ele associada é inserida no conjunto C.

Desta forma, três operações são realizadas na pilha:

1. Inserir o índice correspondente à variável;

2. Alterar o índice do topo para negativo;

3. Retirar o elemento da pilha.

A primeira operação representa a associação a um nó da árvore de busca da variável cujo índice foi inserido na pilha. A primeira modificação nos conjuntos C e J indica que será primeiramente explorada a sub-árvore direita. A segunda operação na pilha, juntamente com a segunda modificação nos conjuntos C e J, indica que a sub-árvore esquerda será explorada, sendo que, depois desta exploração, todas as soluções descendentes já foram calculadas. A segunda modificação da pilha é realizada pelas operações de retirar o elemento e inseri-lo na pilha após ter sido multiplicado por -1. A terceira operação na pilha, juntamente com a terceira modificação nos conjuntos C e J, indica *backtracking*.

O primeiro passo do algoritmo de enumeração é efetuar um pré-processamento no problema, sendo que este pré-processamento identificará as variáveis que pertencerão aos conjuntos N e P, não sendo necessário efetuar o *backtracking* nos nós que estão associados às variáveis pertencentes a estes conjuntos. Como este é o primeiro passo do algoritmo, os nós que estiverem associados às variáveis pertencentes a N e P ficarão em um nível inferior da árvore de busca, em relação a todos os outros nós que estiverem associados a variáveis que não pertencerem a estes conjuntos. Caso haja variáveis em N ou P, uma delas será associada à raiz da árvore de busca. Assim sendo, quando houver um *backtracking* que conduza a um nó v, cuja variável pertença a N ou P, todos os nós acima de v não precisam ser enumerados, em razão de estarem associados às variáveis nestes conjuntos, o que provocaria *backtracking* sucessivo até a raiz, implicando assim na parada do algoritmo.

Após este primeiro passo, deve-se verificar se a condição de parada obteve sucesso. Caso a resposta seja positiva, termina-se a execução do algoritmo, onde a solução ótima é aquela que fora encontrada no pré-processamento. Em caso contrário, começa-se a enumeração, onde cada variável associada a um nó tem seu índice inserido na pilha, fixando-se seu valor em 1.

Se houver *backtracking* em um nó, deve-se verificar se o elemento do topo da pilha é positivo. Se for verdade, altera-se seu valor para negativo, indicando-se que o valor foi fixado em 0. Em caso contrário, isto significa que as duas sub-árvores

já foram exploradas, retirando-se o elemento da pilha e fazendo-se novamente o *backtracking*.

Estas operações são realizadas até que a pilha esteja vazia, indicando a parada do algoritmo, isto é, o *backtracking* na raiz.

Ainda no pré-processamento, $\hat{Z}$ é inicializado e, a cada novo nó explorado, verifica-se se a solução é melhor ou pior. Caso seja melhor, $\hat{Z}$ passará a ter o valor da solução do nó referenciado.

3.3 Algoritmo de Balas

A idéia do algoritmo de Balas [Ba65, Ma77, SyDeKo83, Zi74] será aplicada em uma situação onde todas as soluções são viáveis. Em razão de as variáveis poderem apenas assumir os valores 0 ou 1, o conjunto de todas as soluções é finito e o número de combinações é 2^n, onde n é o número de variáveis x_j.

Portanto, o que o algoritmo faz é considerar uma série de regras segundo as quais pode-se obter uma solução ótima (se tal solução existir), acompanhando algumas ramificações de uma árvore de busca e abandonando muitas outras. Partindo de uma situação na qual todas as n variáveis são iguais a 0, o algoritmo consiste de um procedimento sistemático de atribuir o valor 1 a algumas das variáveis de tal modo que, após testar uma parte pequena de todas as combinações, pode-se obter uma solução ótima ou uma evidência de que não existe solução viável. Desta forma, determina-se em cada iteração:

(a) um subconjunto de variáveis que são candidatas a assumir o valor 1;

(b) a variável a ser escolhida entre as candidatas.

Em alguns níveis deste processo, obtém-se uma solução ótima ou nenhuma solução ótima com valor 1 para todas as variáveis que assumiram este valor. Então, o processo pára e recomeça a partir do nível anterior. Em outras palavras, as regras do algoritmo identificam tais ramificações da árvore de busca, que podem ser abandonadas por não poderem conduzir a uma solução viável melhor do que aquelas já obtidas.

O algoritmo abaixo é baseado no algoritmo de Balas [Ba65] e está descrito da seguinte forma:

início

prePocessamento();

if C = vazio ou $Z'_v \geq \hat{Z}$ **então**

acabou();

índice := associaVariável();

$C = C$ - {índice};

$J = J \cup$ {índice};

inserePilha(índice);

enquanto pilha != vazia **fazer**

if $\hat{Z} >$ soluçãoNóAtual() **então**

$\hat{Z}$:= soluçãoNóAtual();

if C = vazio ou $Z'_v \geq \hat{Z}$ **então**

elemento = retiraPilha();

enquanto elemento < 0 **fazer**

$C = C \cup$ elemento;

elemento = retiraPilha();

if pilha != vazia **então**

inserePilha(-elemento);

$J = J \cup$ elemento;

else

índice := associaVariável();

$C = C$ - índice;

$J = J \cup$ índice;

inserePilha(índice);

fim

- A rotina prePocessamento já foi anteriormente descrita.

- A rotina `acabou` indica o final da enumeração, isto é, o final do algoritmo.
- A rotina `associaVariável` representa qualquer uma das heurísticas de associação de uma variável a um nó.
- A rotina `inserePilha` insere um elemento na pilha.
- A rotina `retiraPilha` retira o elemento da pilha, devolvendo seu valor.
- A rotina `soluçãoNóAtual` devolve o valor da resolução do problema para o nó em questão, isto é, fornece o valor considerando-se os conjuntos associados ao nó.

Capítulo 4

Resultados Computacionais

Neste Capítulo, relatamos os experimentos computacionais obtidos com o Algoritmo de *Branch and Bound* descrito no Capítulo 3. Os testes realizados tinham, como intuito, avaliar a eficiência de tal algoritmo na resolução de problemas de programação linear inteira mista.

O Algoritmo de *Branch and Bound* para problemas de programação linear inteira mista foi implementado em linguagem C++, utilizando um compilador g++ no sistema operacional Debian Linux Versão 2.2.17. Os testes computacionais foram executados em uma máquina dedicada com processador AMD-K6 de 500 MHz, 64 MB de memória RAM e 64 kB de memória cache.

Os tempos de CPU reportados nesta Seção foram obtidos fazendo-se uso da rotina `time()` do Sistema Operacional.

Inicialmente, descreveremos as características das instâncias que foram testadas e, logo em seguida, exibiremos os resultados.

4.1 Descrição das Instâncias

Antes de apresentarmos os problemas-testes, detalharemos o gerador de testes para que possamos ter uma familiarização com a nomenclatura adotada.

As instâncias utilizadas nos experimentos computacionais foram geradas de maneira aleatória, seguindo alguns critérios para sua formação, os quais podem ser

enumerados da seguinte forma:

Gerador {NM} {PN} {E} {N}, onde:

- NM → Número Máximo, que representa o valor máximo, em módulo, que um coeficiente pode assumir, isto é, os coeficientes podem assumir valores no intervalo [-NM, NM];
- PN → Porcentagem de Negativos, que indica a porcentagem de coeficientes negativos para o problema;
- E → Número de Expressões;
- N → Número de Variáveis;
- Ordem da matriz → Número de Expressões x Número de Variáveis.

Os problemas utilizados para a avaliação de desempenho do algoritmo implementado estão divididos em três classes principais:

1. A primeira, denominada Tipo I, abrange instâncias com matrizes de ordem 10 x 10, 20 x 20 e 30 x 30;
2. A segunda, caracterizada por uma instância com matriz de ordem 40 x 40, é chamada de Tipo II;
3. A terceira, chamada de Tipo III, é caracterizada por uma matriz de ordem 50 x 50.

Esclarecemos que os problemas foram gerados aleatoriamente e que a ordem das matrizes pode ser distinta, mas adotamos estes valores para nossos testes.

As tabelas 4.1 a 4.3 relacionam os dados de todas as classes das instâncias testadas, mostrando: na primeira coluna, o nome da instância; na segunda coluna, o número de expressões; na terceira coluna, o número de variáveis; na quarta coluna, a porcentagem de negativos; e na quinta coluna, o número máximo, em módulo, que o coeficiente pode assumir.

Podemos ainda destacar que o nome da instância é definido da seguinte forma: i{número de expressões}x{número de variáveis}x{porcentagem de coeficientes negativos}_{número máximo que um coeficiente pode assumir}.

Instâncias	Expressões	Variáveis	PN	NM
i10x10x30_100	10	10	30	100
i10x10x50_100	10	10	50	100
i20x20x30_100	20	20	30	100
i20x20x50_100	20	20	50	100
i30x30x30_100	30	30	30	100
i30x30x50_100	30	30	50	100
i10x10x30_150	10	10	30	150
i10x10x50_150	10	10	50	150
i20x20x30_150	20	20	30	150
i20x20x50_150	20	20	50	150
i30x30x30_150	30	30	30	150
i30x30x50_150	30	30	50	150
i10x10x30_200	10	10	30	200
i10x10x50_200	10	10	50	200
i20x20x30_200	20	20	30	200
i20x20x50_200	20	20	50	200
i30x30x30_200	30	30	30	200
i30x30x50_200	30	30	50	200

Tabela 4.1: Descrição das instâncias do tipo I

Instâncias	Expressões	Variáveis	PN	NM
i40x40x30_100	40	40	30	100
i40x40x50_100	40	40	50	100
i40x40x30_150	40	40	30	150
i40x40x50_150	40	40	50	150
i40x40x30_200	40	40	30	200
i40x40x50_200	40	40	50	200

Tabela 4.2: Descrição das instâncias do tipo II

Instâncias	Expressões	Variáveis	PN	NM
i50x50x30_100	50	50	30	100
i50x50x50_100	50	50	50	100
i50x50x30_150	50	50	30	150
i50x50x50_150	50	50	50	150
i50x50x30_200	50	50	30	200
i50x50x50_200	50	50	50	200

Tabela 4.3: Descrição das instâncias do tipo III

4.2 Resultados Obtidos

A seguir, apresentaremos as tabelas com os resultados de nossos testes. Tais tabelas apresentam o mesmo formato, como pode ser descrito: a primeira coluna contém o nome da instância testada, de acordo com as tabelas 4.1 a 4.3; a segunda coluna contém o nome da heurística que foi utilizada; a terceira coluna determina o número de nós explorados; a quarta coluna exibe quanto tempo foi consumido por cada instância, de acordo com a heurística aplicada, estando expresso em

segundos e com precisão de duas casas decimais; e a quinta coluna contém o valor da solução ótima ($\hat{Z}$). Uma entrada "—" nas tabelas assinala que o algoritmo não pôde resolver o problema em um tempo inferior a 180 minutos, significando que a execução do algoritmo foi interrompida por ter excedido este limite de tempo, o qual foi estipulado em razão de, para as instâncias utilizadas em nossos testes, ser tempo suficiente para que seja encontrada uma solução ótima.

Instâncias do Tipo I:

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i10x10x30_100	PrimeiraVariável	20	0,01	95
	MínimoMáximo	12	0,00	
	MenorSomaColuna	22	0,01	
	MaiorNegativoColuna	27	0,02	
	MáximoSomaLinha	26	0,01	
i10x10x50_100	PrimeiraVariável	64	0,01	18
	MínimoMáximo	20	0,00	
	MenorSomaColuna	36	0,01	
	MaiorNegativoColuna	78	0,02	
	MáximoSomaLinha	119	0,00	

Tabela 4.4a: Instâncias do Tipo I (10 x 10) e NM = 100

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i10x10x30_150	PrimeiraVariável	7	0,01	148
	MínimoMáximo	2	0,02	
	MenorSomaColuna	4	0,20	
	MaiorNegativoColuna	4	0,10	
	MáximoSomaLinha	11	0,20	
i10x10x50_150	PrimeiraVariável	57	0,02	-7
	MínimoMáximo	34	0,01	
	MenorSomaColuna	31	0,01	
	MaiorNegativoColuna	30	0,02	
	MáximoSomaLinha	85	0,02	

Tabela 4.4b: Instâncias do Tipo I (10 x 10) e NM = 150

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i10x10x30_200	PrimeiraVariável	30	0,01	188
	MínimoMáximo	6	0,00	
	MenorSomaColuna	15	0,01	
	MaiorNegativoColuna	19	0,02	
	MáximoSomaLinha	53	0,01	
i10x10x50_200	PrimeiraVariável	103	0,01	-3
	MínimoMáximo	53	0,00	
	MenorSomaColuna	91	0,01	
	MaiorNegativoColuna	91	0,01	
	MáximoSomaLinha	144	0,00	

Tabela 4.4c: Instâncias do Tipo I (10 x 10) e NM = 200

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i20x20x30_100	PrimeiraVariável	157	0,01	97
	MínimoMáximo	43	0,01	
	MenorSomaColuna	156	0,02	
	MaiorNegativoColuna	134	0,02	
	MáximoSomaLinha	247	0,02	
i20x20x50_100	PrimeiraVariável	7918	0,22	67
	MínimoMáximo	2685	0,10	
	MenorSomaColuna	5324	0,18	
	MaiorNegativoColuna	4649	0,16	
	MáximoSomaLinha	12574	0,43	

Tabela 4.5a: Instâncias do Tipo I (20 x 20) e NM = 100

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i20x20x30_150	PrimeiraVariável	201	0,03	128
	MínimoMáximo	60	0,01	
	MenorSomaColuna	172	0,02	
	MaiorNegativoColuna	161	0,01	
	MáximoSomaLinha	298	0,03	
i20x20x50_150	PrimeiraVariável	6247	0,18	41
	MínimoMáximo	3028	0,11	
	MenorSomaColuna	4572	0,15	
	MaiorNegativoColuna	4482	0,14	
	MáximoSomaLinha	8701	0,29	

Tabela 4.5b: Instâncias do Tipo I (20 x 20) e NM = 150

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i20x20x30_200	PrimeiraVariável	572	0,03	192
	MínimoMáximo	248	0,03	
	MenorSomaColuna	378	0,02	
	MaiorNegativoColuna	347	0,02	
	MáximoSomaLinha	925	0,04	
i20x20x50_200	PrimeiraVariável	11120	0,32	92
	MínimoMáximo	3018	0,11	
	MenorSomaColuna	4153	0,15	
	MaiorNegativoColuna	3900	0,13	
	MáximoSomaLinha	10644	0,35	

Tabela 4.5c: Instâncias do Tipo I (20 x 20) e NM = 200

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i30x30x30_100	PrimeiraVariável	1537	0,06	96
	MínimoMáximo	284	0,02	
	MenorSomaColuna	1282	0,06	
	MaiorNegativoColuna	1474	0,07	
	MáximoSomaLinha	868	0,04	
i30x30x50_100	PrimeiraVariável	>100000	23,35	69
	MínimoMáximo	>100000	8,60	
	MenorSomaColuna	>100000	15,42	
	MaiorNegativoColuna	>100000	12,02	
	MáximoSomaLinha	>100000	46,25	

Tabela 4.6a: Instâncias do Tipo I (30 x 30) e NM = 100

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i30x30x30_150	PrimeiraVariável	5744	0,20	147
	MínimoMáximo	854	0,06	
	MenorSomaColuna	1863	0,09	
	MaiorNegativoColuna	2661	0,13	
	MáximoSomaLinha	2750	0,13	
i30x30x50_150	PrimeiraVariável	>100000	34,87	42
	MínimoMáximo	>100000	12,33	
	MenorSomaColuna	>100000	12,40	
	MaiorNegativoColuna	>100000	14,44	
	MáximoSomaLinha	>100000	20,46	

Tabela 4.6b: Instâncias do Tipo I (30 x 30) e NM = 150

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i30x30x30_200	PrimeiraVariável	9945	0,33	195
	MínimoMáximo	1816	0,09	
	MenorSomaColuna	13148	0,55	
	MaiorNegativoColuna	8123	0,33	
	MáximoSomaLinha	16281	0,74	
i30x30x50_200	PrimeiraVariável	>100000	42,81	77
	MínimoMáximo	>100000	13,95	
	MenorSomaColuna	>100000	18,76	
	MaiorNegativoColuna	>100000	19,14	
	MáximoSomaLinha	>100000	60,46	

Tabela 4.6c: Instâncias do Tipo I (30 x 30) e NM = 200

Instâncias do Tipo II:

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i40x40x30_100	PrimeiraVariável	54733	2,04	99
	MínimoMáximo	6989	0,37	
	MenorSomaColuna	45842	2,32	
	MaiorNegativoColuna	27831	1,39	
	MáximoSomaLinha	45296	2,44	
i40x40x50_100	PrimeiraVariável	>100000	6696,24	99
	MínimoMáximo	>100000	1589,21	
	MenorSomaColuna	>100000	9233,90	
	MaiorNegativoColuna	>100000	7035,04	
	MáximoSomaLinha	>100000	9707,13	

Tabela 4.7a: Instâncias do Tipo II (40 x 40) e NM = 100

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i40x40x30_150	PrimeiraVariável	74813	2,80	148
	MínimoMáximo	8879	0,48	
	MenorSomaColuna	17470	0,91	
	MaiorNegativoColuna	21836	1,09	
	MáximoSomaLinha	81315	4,38	
i40x40x50_150	PrimeiraVariável	>100000	—	64
	MínimoMáximo	>100000	2704,46	
	MenorSomaColuna	>100000	4257,99	
	MaiorNegativoColuna	>100000	4393,89	
	MáximoSomaLinha	>100000	—	

Tabela 4.7b: Instâncias do Tipo II (40 x 40) e NM = 150

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i40x40x30_200	PrimeiraVariável	>100000	7,01	198
	MínimoMáximo	27293	1,41	
	MenorSomaColuna	95057	4,78	
	MaiorNegativoColuna	95308	4,65	
	MáximoSomaLinha	>100000	9,67	
i40x40x50_200	PrimeiraVariável	>100000	—	96
	MínimoMáximo	>100000	4070,33	
	MenorSomaColuna	>100000	5308,76	
	MaiorNegativoColuna	>100000	7785,89	
	MáximoSomaLinha	>100000	—	

Tabela 4.7c: Instâncias do Tipo II (40 x 40) e NM = 200

Instâncias do Tipo III:

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i50x50x30_100	PrimeiraVariável	>100000	21,26	95
	MínimoMáximo	40641	2,49	
	MenorSomaColuna	>100000	33,14	
	MaiorNegativoColuna	>100000	24,92	
	MáximoSomaLinha	>100000	51,49	
i50x50x50_100	PrimeiraVariável	>100000	—	94
	MínimoMáximo	>100000	—	
	MenorSomaColuna	>100000	—	
	MaiorNegativoColuna	>100000	—	
	MáximoSomaLinha	>100000	—	

Tabela 4.8a: Instâncias do Tipo III (50 x 50) e NM = 100

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i50x50x30_150	PrimeiraVariável	>100000	146,17	149
	MínimoMáximo	>100000	26,35	
	MenorSomaColuna	>100000	191,32	
	MaiorNegativoColuna	>100000	104,36	
	MáximoSomaLinha	>100000	180,01	
i50x50x50_150	PrimeiraVariável	>100000	—	49
	MínimoMáximo	>100000	—	
	MenorSomaColuna	>100000	—	
	MaiorNegativoColuna	>100000	—	
	MáximoSomaLinha	>100000	—	

Tabela 4.8b: Instâncias do Tipo III (50 x 50) e NM = 150

Instâncias	Heurísticas	Nós	Tempo (seg)	$\hat{Z}$
i50x50x30_200	PrimeiraVariável	>100000	70,63	197
	MínimoMáximo	>100000	68,50	
	MenorSomaColuna	>100000	91,00	
	MaiorNegativoColuna	>100000	71,93	
	MáximoSomaLinha	>100000	173,98	
i50x50x50_200	PrimeiraVariável	>100000	—	116
	MínimoMáximo	>100000	—	
	MenorSomaColuna	>100000	—	
	MaiorNegativoColuna	>100000	—	
	MáximoSomaLinha	>100000	—	

Tabela 4.8c: Instâncias do Tipo III (50 x 50) e NM = 200

Capítulo 5

Conclusão

O objetivo deste trabalho foi desenvolver heurísticas que apresentassem a solução ótima para um problema de programação linear inteira mista, onde todas as soluções são viáveis, após ser explorado o menor número possível de soluções e em um tempo computacional mínimo.

Destacamos a seguir uma análise sobre os resultados computacionais obtidos com nossos experimentos, ressaltando que os melhores resultados foram obtidos através da análise do menor número de nós e em um tempo computacional menor.

5.1 Instâncias do Tipo I

Nas instâncias do tipo I e com matriz de ordem 10 x 10, dentre os resultados obtidos, a heurística **MínimoMáximo** se destaca em todas as instâncias, onde o algoritmo encontrou a solução ótima de todas estas instâncias com um tempo bem abaixo do estabelecido e explorando um número pequeno de nós.

Uma outra heurística que também se destacou foi a heurística **MenorSoma-Coluna**, a qual foi semelhante à heurística **MínimoMáximo**, porém forneceu um número maior de nós explorados e em um tempo computacional um pouco mais elevado.

Observa-se também que, em alguns casos, o número de nós explorados foi maior do que 100 nas instâncias i10x10x50_100 e i10x10x50_200 (esta em duas heurísti-

cas), mas o tempo computacional se manteve na média dos resultados obtidos nos outros testes.

Nos testes das instâncias do tipo I, com matrizes de ordem 20 x 20 e 30 x 30, as avaliações das heurísticas foram mantidas. Porém, houve um aumento considerável no número de nós explorados e em um tempo computacional semelhante às outras instâncias desta classe.

5.2 Instâncias do Tipo II

As instâncias do tipo II mostraram um panorama diferente daquelas do tipo I. Enquanto nos testes das instâncias do tipo I o tempo de execução do algoritmo se manteve na média, nas instâncias do tipo II houve um aumento elevado, chegando, em alguns casos isolados, ao término de sua execução de forma prematura, em razão de o limite de tempo estabelecido ter sido ultrapassado, inclusive com um número mais elevado de nós explorados.

Porém, as avaliações das heurísticas foram mantidas, sendo a de melhor desempenho a heurística **MínimoMáximo**, seguida pela heurística **MenorSomaColuna**.

5.3 Instâncias do Tipo III

Nas instâncias do tipo III, destacaram-se as heurísticas **MínimoMáximo** e **MaiorNegativoColuna**, por terem obtido resultados melhores no tempo computacional e no número de nós explorados. Pode-se ainda destacar que, para instâncias com a porcentagem de coeficientes negativos de 50%, os testes foram interrompidos, em razão de o tempo computacional apresentado ter sido superior ao estipulado.

As demais heurísticas, denominadas **PrimeiraVariável** e **MáximoSomaLinha**, obtiveram resultados semelhantes na classe do tipo I das instâncias pequenas, mas, em relação às classes dos tipos II e III, não obtiveram valores satisfatórios de tempo computacional razoável e de número de nós explorados na árvore

de *Branch and Bound*. Assim sendo, não foram comentadas nos itens anteriores.

De acordo com os resultados obtidos e exibidos no capítulo anterior, podemos afirmar que ainda existe campo para pesquisa, além de oportunidades para serem exploradas na área de algoritmos para programação linear inteira mista, pois ressaltamos a escassez de referências bibliográficas sobre este tema.

Referências Bibliográficas

[Ba65] BALAS, E. "An additive algorithm for solving linear programs with zero-one variables", *Operations Research* v. 13, pp. 517-546, 1965.

[BaJaSh90] BAZARAA, M. S., JARVIS, J. J., SHERALI, H. D., 1990, *Linear Programming and Network Flows.* New York, John Wiley and Sons.

[Be62] BENDERS, J. F., "Partitioning Procedures for Solving Mixed Variables Programming Problems", *Numerische Mathematics*, v. 4, pp. 238-252, 1962.

[Ga94] GASS, S. I., *Linear Programming.* New York, McGraw-Hill, Inc.

[GoLu2000] GOLDBARG, M. C., LUNA, H. P. L., 2000, *Otimização Combinatória e Programação Linear.* Rio de Janeiro, Editora Campus Ltda.

[Gr75] GREENBERG, H., 1975, *Integer Programming.* New York, Academic Press.

[MaPe80] MACULAN FILHO, N., PEREIRA, M. V. F., 1980, *Programação Linear.* São Paulo, Atlas.

[Ma77] MACULAN FILHO, N., 1977, *Programação Linear Inteira.* Rio de Janeiro, COPPE / UFRJ.

[Mi86] MINOUX, M., 1986, *Mathematical Programming. Theory and Algorithms.* New York, John Wiley and Sons.

[NeWo88] NEMHAUSER, G. L., WOLSEY, L. A., 1988, *Integer and Combinatorial Optimization.* New York, John Wiley and Sons.

[PaRa88] PARKER, R. G., RARDIN, R. L., 1988, *Discrete Optimization.* Londres, Academic Press, Inc.

[SaKa89] SALKIN, H. M., KAMLESH, M., 1989, *Foundations of Integer Programming.* New York, North-Holland.

[Sa75] SALKIN, H. M., 1975, *Integer Programming.* Cleveland, Addison-Wesley Publishing Company.

[SyDeKo83] SYSLO, M. M., DEO, N., KOWALIK, J. S., 1983, *Discrete Optimization Algorithms.* New Jersey, Prentice-Hall, Inc.

[Ta75] TAHA, H. A., 1975, *Integer Programming - theory, applications, and computations.* New York, Academic Press, Inc.

[Wo98] WOLSEY, L. A., 1998, *Integer Programming.* New York, John Wiley and Sons.

[Zi74] ZIONTS, S., 1974, *Linear and Integer Programming.* New Jersey, Prentice-Hall, Inc.