CONTRIBUIÇÃO ÀS APLICAÇÕES DOS MODELOS DE RECOBRIMENTO E PARTICIONAMENTO EM PROGRAMAÇÃO INTEIRA

Geraldo Galdino de Paula Junior

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 (M.Sc.).

Aprovada por:

Prof. Nelson Maculan Filho
(Presidente)

Prof. João Lizardo R.H. de Araújo

Prof. Demétrio Alonso Ribeiro

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 1978

PAULA JUNIOR, GERALDO GALDINO DE

Contribuição às Aplicações dos Modelos de Recobrimento e Particionamento em Programação Inteira [Rio de Janeiro] 1978.

IX, 160p. 29,7cm (COPPE-UFRJ, M.Sc. Engenharia de Sistemas e Computação, 1978)

Tese - Univ. Fed. Rio de Janeiro. Prog. Engenharia de Sistemas e Computação

1. Otimização 2. Pesquisa Operacional I. COPPE/UFRJ II. Título(Série).

A Maria do Carmo

Ao Dudu

A meus pais e irmãs

Agradecimentos

Várias pessoas e instituições colaboraram para a realização deste trabalho. A elas agradecemos e em especial

ao Professor Nelson Maculan, pelo seu incentivo constante e pela orientação eficiente, colocando a nossa disposição seu amplo cabedal de experiência e conhecimentos técnicos;

ao Professor Michel Minoux, da Ecole Nationale Supérieur de Techniques Avancée, de Paris, com quem tivemos a oportunidade de discutir importantes aspectos deste trabalho, quando de sua passagem pela COPPE;

à Universidade Federal de Viçosa, que proporcionou suporte financeiro para a realização desta pesquisa;

aos Professores João Lizardo R. H. de Araújo e Demétrio Alonso Ribeiro, pela participação da banca de Tese;

e à Coordenação dos Programas de Pós-graduação de Engenharia da Universidade Federal do Rio de Janeiro, pela oportunidade do Mestrado em Engenharia de Sistemas e Computação.

Sumário

Este trabalho propõe um código para a solução automática de problemas combinatórios em Programação Inteira bivalente. O algoritmo de Enumeração Implícita é utilizado na forma proposta por GEOFFRION[22], como simplificação do algoritmo de BALAS[2]. Restrições substitutas ou Filtros, são implementados para acelerar o processo de enumeração de pseudo-soluções parciais. Os problemas de determinação dos conjuntos Recobrimento e Particionamento mínimos e suas formas particulares dentro da teoria dos grafos não orientados são tratados aqui, e como aplicação, é proposta uma metodologia para a solução do problema de alocação ótima ou pelo menos viável, de tripulações em rotas aéreas de uma companhia de aviação.

Summary

This work proposes a code for automatic solution of combinatorial problems in bivalent Integer Programming. The Implicit Enumeration algorithm is utilized in the form proposed by GEOFFRION[22], as a simplification of the BALAS[2] algorithm. Filters or surrogate constraints are implemented to accelerate the partial pseudo-solutions enumeration process. The set-covering and set-partitioning problems and its particular versions within non-oriented graph theory, are treated here. As an application, a methodology is proposed for the solution of airline crew scheduling problem.

I N D I C E

CAPÍTULOS PÁGINAS

CAPÍTULOS PÁGINAS

CAPÍTULOS PÁGINAS

CAPÍTULO I

MODELOS COMBINATÓRIOS: SUAS ÁREAS DE APLICAÇÃO, HISTÓRICO, DIFICULDADE COMPUTACIONAL E PLANO DE TRABALHO.

1.1 - INTRODUÇÃO.

Na literatura corrente sobre Programação Matemática e em particular sobre Programação Inteira, encontramos fartas referências ao fato de que o potencial de aplicação desta área tem crescido enormemente e nas duas últimas décadas numerosos avanços teóricos foram registrados. Como resultado destes avanços temos hoje uma vasta coleção de métodos e algoritmos que aliados ao desenvolvimento da exatidão, velocidade e sofisticação dos sistemas de computadores digitais, prometem ser de grande valia na solução de importantes problemas práticos que estão surgindo.

Neste trabalho tentaremos fazer ligeiras contribuições às aplicações de resultados de natureza combinatória, colocando à disposição um código para solução dos problemas de Programação Inteira bivalente, de determinação dos conjuntos recobrimento e particionamento mínimos, emparelhamento máximo e recobrimento mínimo em grafos não orientados.

Desde que as circunstâncias forçaram uma opção pela sofisticação tecnológica, vários problemas de natureza combinatória surgiram e continuam aparecendo e os identificamos frequentemente na construção de grandes redes de transmissão de informações ou de dados para processamento automático, no setor de transporte urbano, na alocação de tripulações e aviões das companhias aéreas, na administração do espaço físico em escolas e universidades, no setor político, na construção de circuitos eletrônicos impressos, na recuperação de informações de grandes arquivos em centros de processamento de dados, na seleção de projetos sujeitos a uma restrição or-

çamentária, etc. Com a necessidade das soluções desses problemas, está surgindo uma área de estudos que poderiamos denominar de "otimização combinatória" e é nesse campo que pretendemos desenvolver algumas experiências.

1.2 - HISTÓRICO.

Naturalmente foram muitos os pesquisadores que trabalharam com processos enumerativos de busca de soluções de problemas de Programação Inteira, que promoveram progressos no estudo de situações especialmente estruturadas, como os problemas dos conjuntos recobrimento e particionamento, suas relações e seu enfoque particular dentro da teoria dos grafos. Mas para fins do nosso trabalho e seguindo a ordem em que abordaremos os fatos, faremos um breve histórico, relacionando somente os nomes dos principais contribuintes.

Praticamente, o primeiro trabalho a estabelecer valiosos critérios no sentido de otimizar processos de enumeração e a receber ampla divulgação nos setores especializados, foi o de LAND e DOIG em 1960, resumidamente referenciado pelo nome de "Branch and Bound". Posteriormente, em 1963, BALAS introduziu as bases da enumeração implícita bivalente, aperfeiçoada por ele mesmo em 1965, quando o processo chamou a atenção dos especialistas de Pesquisa Operacional. Seguindo a esta fase, GLOVER, a partir de 1965, introduziu sensíveis progressos, mostrando a possibilidade de serem usadas restrições especiais que funcionariam como filtros, promovendo a aceleração do processo. GEOFFRION, 1967 e 1969, mostrou um interessante processo de gerar esses filtros a partir de soluções produzidas por variáveis duais relativas às restrições do problema proposto, juntamente com uma ligeira modificação do conceito de "poder" desses filtros no sentido de acelerar a convergência, inicialmente proposto por GLOVER. Daí em diante e por algum tempo, BALAS, GLOVER e GEOFFRION se alternaram e outros pesquisadores como SPIELBERG, LEMKE e SALKIN, introduziram refinamentos valiosos na forma dos

testes existentes. GARFINKEL e NEMHAUSER, promoveram estudos e algoritmos especiais para a solução dos problemas de determinação dos conjuntos recobrimento e particionamento mínimos. Nesse sentido podemos citar mais uma vez a participação de BALAS que proporcionou, juntamente com PADBERG, resultados interessantes nesse campo. A equivalência entre o emparelhamento máximo e o recobrimento mínimo em um grafo não orientado, foi construida por NORMAN e RABIN em 1959. Um recente trabalho de BALAS e SAMUELSSON, 1977, apresentou um novo algoritmo para os problemas de emparelhamento e recobrimento em grafos não orientados. Outros pesquisadores, como GREENBERG, BELLMORE, SPITZER, BALINSKI e QUANDT, se ocuparam de aplicações, resolvendo importantes problemas práticos, utilizando os recursos combinatórios mencionados.

1.3 - DIFICULDADES DA ÁREA E POLÍTICA PARA VENCÊ-LAS.

Diferentemente ao caso dos programas contínuos (Programação Linear), não há esperanças de, no futuro próximo, podermos resolver todos os problemas de Programação Inteira por um único algoritmo. Por outro lado, a dificuldade computacional observada nas técnicas de Programação Inteira de caráter geral, conduziu à formulação e desenvolvimento de métodos especiais para solução de problemas inteiros que possuam estruturas particulares. Isto é, algoritmos especiais são continuamente produzidos utilizando a estrutura particular de uma família de problemas. Uma outra possibilidade, é projetar algoritmos complexos, capazes de analisar, avaliar e decidir que tipo de procedimento poderia ser aplicado na solução do problema particular que se apresentasse. Por exemplo, existem algoritmos para solução do problema de determinação do conjunto recobrimento mínimo, que trabalham melhor com matrizes de alta densidade e outros, ao contrário, são mais eficazes no trato com matrizes de baixa densidade. Aceitamos a idéia de que qualquer estudo experimental dentro deste campo, deva ser levado a efeito com uma polí

tica (que pode ser uma dessas), bem definida em mente, o que tenderia minimizar dificuldades de ordem global.

1.4 - PLANO DE TRABALHO.

Iniciaremos com um capítulo sobre enumeração parcial que deverá se constituir no miolo de todo o processo que produzirá na sua última forma, a ferramenta com que poderemos resolver o problema (aplicação) proposto no CAPÍTULO VI. Em seguida cuidaremos de uma parte dedicada à questão da determinação dos conjuntos recobrimento e particionamento mínimos (na sua forma prática, o modelo de particionamento se assemelha ao de recobrimento com restrições de igualdade). Aí, no CAPÍTULO III, estará o esqueleto da aplicação que pretendemos recomendar. Segue o CAPÍTULO IV, onde relacionamos as idéias particulares, daquelas do CAPÍTULO III, dentro da Teoria dos Grafos, i. e., trataremos com o emparelhamento máximo e recobrimento mínimo em grafos não orientados. No CAPÍTULO V, descrevemos a construção do código, objeto central de nossas experiências e finalmente, no CAPÍTULO VI, ilustramos a aplicabilidade do programa, recomendando uma aplicação referente à alocação de tripulações em rotas aéreas para uma companhia de aviação.

CAPÍTULO II

O PROCESSO DE ENUMERAÇÃO DAS PSEUDO-SOLUÇÕES.

2.1 - INTRODUÇÃO.

Vamos colecionar e ordenar os resultados necessários para a solução de um problema de Programação Inteira bivalente, pelo método de enumeração parcial ou enumeração implícita, como é conhecido na literatura corrente.

2.2 - DEFINIÇÃO DO PROBLEMA.

Seja z uma função

$$z : \{0,1\}^n \to K \subset R \tag{II.1}$$

$$z(x) = \sum_{j \in N} c_j x_j, \qquad N = \{1,\ldots,n\} \tag{II.2}$$

Sobre z queremos resolver o seguinte problema:

$$(P) \qquad \text{minimizar } z = \sum_{j \in N} c_j x_j \tag{II.3}$$

$$\text{sujeito a } b_i + \sum_{j \in N} a_{ij} x_j \geq 0, \quad i \in M = \{1, \ldots, m\} \tag{II.4}$$

$$x_j \in \{0,1\} \qquad j \in N. \tag{II.5}$$

Algumas vezes preferiremos uma forma mais compacta:

$$\text{minimizar } z = cx \tag{II.6}$$

$$\text{sujeito a } b + Ax \geq \underline{0}, \tag{II.7}$$

onde c é o vetor de n dimensões $(c_1, \ldots, c_n)$, b e $\underline{0}$, respectivamente os vetores de m dimensões $(b_1, \ldots, b_m)^t$ e $(0, \ldots, 0)^t$. $x = (x_1, \ldots, x_n)^t$ é um vetor binário de n dimensões e $A = (a_{ij})$, é uma matriz mxn. Qualquer vetor binário x de n dimensões, será chamado uma "pseudo-solução" de (P). Qualquer pseudo-solução que satisfizer (II.4), será chamada de solução viável de (P) e a solução viável de (P) que minimizar z entre todas as soluções viáveis, implícita ou explicitamente enumeradas, será chamada de solução ótima de (P).

Suporemos sempre que $c \geq 0$, sem que isto constitua uma restrição à generalidade, uma vez que, ocorrendo $c_j < 0$, transformaremos x_j em $1 - x'_j$. No decorrer da apresentação das idéias, ficará clara a necessidade de mantermos sempre esta situação.

Também aproveitamos a possibilidade de podermos desenvolver um número na sua correspondente forma (expressão) de base 2, para estendermos nosso algoritmo na tarefa de solucionar qualquer problema de Programação Inteira limitado, i.e., qualquer problema de Programação Inteira onde cada variável $x_k \geq 0$, possui uma cota superior u_k. De fato, se x_k é uma variável inteira qualquer, possuindo uma cota superior u_k, i.e., $x_k \leq u_k$, podemos escrever

$$x_k = \sum_{p=0}^{K} 2^p y_{kp}, \qquad y_{kp} \in \{0,1\} \tag{II.8}$$

onde K é o menor número determinado de tal forma a satisfazer $2^{K+1} - 1 \geqslant u_k$. O inconveniente desta representação, é a grande quantidade de variáveis que surgem para grandes valores de u_k, tornando a solução do problema impraticável dentro de um tempo computacionalmente viável.

2.3 - PROCEDIMENTO PARA ENUMERAR PARCIALMENTE AS SOLUÇÕES DE (P).

Para formalizar melhor as idéias envolvidas neste procedimento, precisamos de alguns termos (ou expressões), cujas definições daremos agora. Comecemos por "pseudo-solução parcial".

Uma pseudo-solução parcial, é uma atribuição de valores binários a um subconjunto das n variáveis.

Uma variável à qual não se atribuiu valor, será chamada de "variável livre" (trata-se de uma variável livre para receber uma atribuição binária no momento em que os testes do algoritmo determinarem).

Representaremos uma pseudo-solução parcial por W, conjunto dos índices das variáveis que compõem a pseudo-solução parcial, de tal forma que se $j \in W$, $x_j=1$ e se $-j \in W$, $x_j=0$. Para ilustrar, se

$$n=6 \qquad \text{e} \qquad W=\{2,4,-6\},$$

teremos $x_2=1$, $x_4=1$, $x_6=0$ e as variáveis livres relativamente a esta pseudo-solução parcial, serão x_1, x_3 e x_5.

Dada uma pseudo-solução parcial W, tal que card(W)=k, existem n-k variáveis livres. Atribuindo valores 0-1 a essas n-k variáveis livres, podemos formar 2^{n-k} vetores de n-k dimensões, que chamaremos de "vetores livres". Tomando o pequeno exemplo dado, temos os seguintes vetores li-

vres referentes a W:

$$(0,0,0),\ (0,1,0),\ (0,1,1),\ (0,0,1),\ (1,0,0),\ (1,1,0),\ (1,1,1),\ (1,0,1).^{*}$$

Com isso podemos definir o complemento de uma pseudo-solução parcial.

Um "complemento" ou "remate" de uma pseudo-solução parcial W, é uma pseudo-solução definida pelos valores das variáveis cujos índices estão em W, juntamente com os valores das variáveis especificados num dos vetores livres. Mais uma vez, se n=6 e $W=\{2,4,-6\}$, o remate da pseudo-solução parcial (?,1,?,1,?,0), é a pseudo-solução (1,1,0,1,1,0), formada com o vetor livre (*) dado acima.

O complemento nulo de uma pseudo-solução parcial, é uma pseudo-solução formada com o vetor livre que tem todos os elementos iguais a zero.

O número de complementos de uma pseudo-solução parcial é naturalmente o mesmo de vetores livres. Portanto, uma pseudo-solução parcial W, tal que card(W)=k, tem 2^{n-k} complementos ou remates. W é o complemento de si mesma quando não existem variáveis livres. Neste caso, card(W)=n.

Na descrição do procedimento para enumerar as pseudo-soluções parciais de (P), as variações dos verbos "podar" e "sondar", terão significados tecnicamente precisos:

O ato de "podar" uma pseudo-solução parcial (P), executado pelo algoritmo, corresponde à ação de:

i) enumerar a pseudo-solução parcial e enumerar implicitamente todos os seus complementos, se a pseudo-solução parcial não tem complemento viável.

ii) caso contrário, enumerar o melhor complemento viável da pseudo-solução parcial e enumerar implicitamente todos os seus outros complementos.

O algoritmo "sonda" uma pseudo-solução parcial quando ela é viável (ou tem complemento viável), verificando se seu melhor complemento viável produz uma cota superior sobre a solução ótima melhor (menor) do que

aquela conhecida até aqui. Em seguida poda a referida pseudo-solução parcial.

Faremos referência ao "um" como complemento lógico do "zero" e vice-versa, o que certamente não será confundido com complemento de uma pseudo-solução parcial.

O número de todos os vetores binários que definimos como pseudo-soluções de (P), é 2^n, e portanto, por um processo exaustivo de enumeração, poderíamos encontrar a solução ótima de (P), quando ela existisse. Entretanto, qualquer busca exaustiva da solução de (P), seria por demais dispendiosa, principalmente em termos de tempo, para os valores de n que surgem nas aplicações práticas. A idéia da enumeração parcial ou implícita, proposta por BALAS,[2] surgiu no sentido de estabelecer um algoritmo munido de critérios que permitissem podar todas as pseudo-soluções parciais sem percorrer exaustivamente a árvore gerada por elas. Esses critérios dão sequência a um mecanismo dinâmico de avanço e retorno sobre os ramos da árvore em cujos nós estão as pseudo-soluções.

O algoritmo estabelece uma poda passando ao complemento lógico do último elemento ainda não complementado na pseudo-solução parcial podada e liberando as variáveis cujos índices figuram na pseudo-solução parcial podada, imediatamente à direita dele.

Em resumo, enumeração implícita é um processo de solução baseado na geração de um grafo tipo árvore, cujo nós são depositários das pseudo-soluções parciais e baseado em critérios que permitem enumerar implícita ou explicitamente todas as pseudo-soluções possíveis.

W é um conjunto ordenado, no sentido de que a ordem dos seus elementos, reflete a ordem em que foram gerados, i. e., reflete a sequência natural definida pelos mecanismos de avanço e retorno do processo enumerativo. Quando um índice de W é logicamente complementado, ele é também sublinhado para indicar que o outro valor de sua variável já foi considerado. Algumas vezes abreviaremos por "complemento lógico", aquilo que na verdade é o complemento lógico sublinhado.

Quando uma pseudo-solução parcial é produzida, o algoritmo

verifica se ela é viável. Se for, verifica ainda (sonda) se seu melhor complemento viável fornece uma cota superior sobre a solução ótima, melhor do que aquela conhecida até aqui. Se isto acontece, a pseudo-solução parcial é considerada sondada, seus complementos são excluidos de futuras considerações, excessão feita de seu melhor complemento viável que é armazenado, juntamente com a cota, para futuras averiguações (futuras comparações). Se a cota produzida não é melhor do que a existente, o algoritmo simplesmente poda a pseudo-solução parcial em questão. Se a pseudo-solução parcial não é viável, o algoritmo tenta vencer a inviabilidade, buscando uma pseudo-solução parcial descendente desta, da seguinte forma: coleciona as variáveis livres que fixadas ajudariam na obtenção de uma cota superior melhor do que a existente e cooperariam para vencer a inviabilidade. Se não existem variáveis livres com essas características, o algoritmo poda a pseudo-solução parcial. Caso contrário, ele elege dentre as variáveis com as características acima, aquela que minimiza a quantidade total de inviabilidade existente. O índice desta variável é acrescentado àqueles da pseudo-solução parcial considerada, formando com eles uma nova pseudo-solução parcial descendente daquela.

Representamos por (W^k) a seqüência das pseudo-soluções parciais. O algoritmo inicia o processo de enumeração com $W^o = \phi$. O processo terminaria nesse ponto se o algoritmo podasse W^o, pois $card(W^o)=0$ e os complementos de W^o, em número de 2^n, seriam implicitamente enumerados. Caso contrário, o algoritmo obtém W^1, como descendente de W^o, pelo aumento desta com a eleição de uma variável livre, a cada passo tentando podar (ainda que depois de sondar), a presente pseudo-solução parcial. Prosseguindo assim, até que na p-ésima pseudo-solução parcial, W^p é sondada. O melhor complemento de W^p, se este produzir uma cota superior sobre a solução ótima melhor do que a cota conhecida até então, é guardado como uma solução candidata.

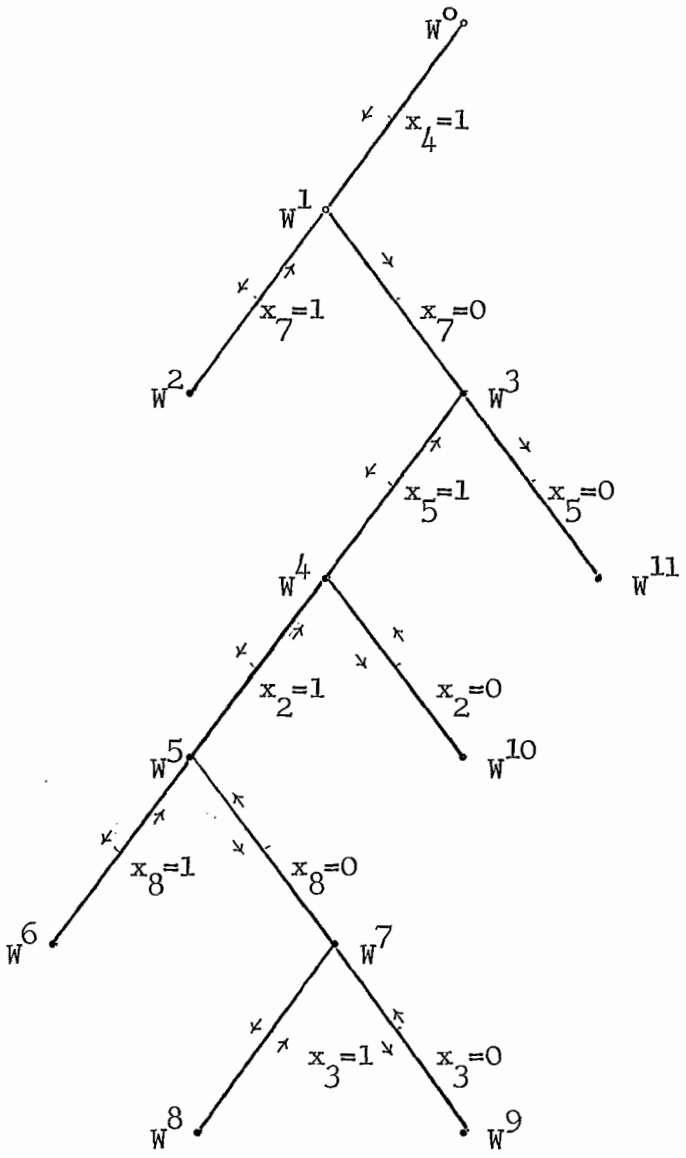
Dois membros sucessivos, W^p e W^{p+1}, da seqüência (W^k) são distintos ou porque $W^p \subset W^{p+1}$, que acontece quando W^{p+1} é descendente de W^p ou porque possuem elementos logicamente complementares. Este fato faz da seqüência (W^k), uma seqüência não redundante, no sentido de que ela não dupli

ca nenhuma pseudo-solução parcial previamente podada.

A fim de visualizar melhor o que foi dito acima, a figura II.1, mostra parte de uma árvore gerada pelas pseudo-soluções parciais de um problema de Programação Inteira bivalente. Lá estão representados doze elementos da sequência (W^k). Através destes elementos podemos observar a lei que rege a formação da sequência de pseudo-soluções parciais. Passamos de um membro para outro da sequência, seja pela complementação lógica do último elemento ainda não complementado, seja pelo acréscimo de mais um elemento. No primeiro caso, abandonamos todos os elementos imediatamente à direita do último elemento complementado. Este fato corresponde a um retorno nos ramos da árvore. Quando a passagem para o próximo membro da sequência, se faz através do acréscimo de mais um elemento, o algoritmo está provocando um avanço nos ramos da árvore (criando um descendente). Observamos que a diferença entre W^1 e W^2, está no elemento a mais que W^2 possui (W^2 é descendente de W^1) e a diferença entre W^2 e W^3, está no elemento logicamente complementado que W^3 possui.

2.4 - CRITÉRIO PARA CONSTRUÇÃO DO MELHOR COMPLEMENTO DE UMA PSEUDO-SOLUÇÃO PARCIAL.

Se para uma pseudo-solução parcial W, for construido um complemento, atribuindo a cada variável livre x_j o valor 0 ou 1, conforme $c_j \geqslant 0$ ou $c_j < 0$, respectivamente e lembrando que nosso problema é de minimização, estaremos diante de um complemento da pseudo-solução parcial, que se for viável, será o melhor complemento viável em questão. Como os custos de (P) são todos positivos ou temos a possibilidade de obtê-los assim, pela transformação de variáveis já explicada no início, observamos que o complemento nulo de uma pseudo-solução parcial, quando for viável, será o melhor complemento viável nessas condições. Assim, se o complemento nulo viável de uma pseudo-solução parcial não produzir uma cota superior sobre a solução



$W^{o} = \phi$

$W^{1} = \{4\}$

$W^{2} = \{4,7\}$

$W^{3} = \{4,-\underline{7}\}$

$W^{4} = \{4,-\underline{7},5\}$

$W^{5} = \{4,-\underline{7},5,2\}$

$W^{6} = \{4,-\underline{7},5,2,8\}$

$W^{7} = \{4,-\underline{7},5,2,-\underline{8}\}$

$W^{8} = \{4,-\underline{7},5,2,-\underline{8},3\}$

$W^{9} = \{4,-\underline{7},5,2,-\underline{8},-\underline{3}\}$

$W^{10} = \{4,-\underline{7},5,-\underline{2}\}$

$W^{11} = \{4,-\underline{7},-\underline{5}\}$

⋮

Figura II.1

Parte de uma árvore de pseudo-soluções parciais de um problema de Programação Inteira bivalente

ótima melhor do que a atual, o algoritmo não tem necessidade de testar os outros complementos e o que ele tem a fazer, é podar a pseudo-solução parcial em questão.

Com relação ao complemento nulo de cada pseudo-solução parcial, o algoritmo tem duas coisas a fazer:

i. Testar sua viabilidade.

ii. Se for viável, verificar (sondar) se a cota superior sobre a solução ótima, gerada por este complemento, é melhor do que a última cota armazenada como candidata ao ótimo de (P)..

Obviamente, a segunda condição será testada se a primeira se cumprir e do cumprimento da segunda condição, resultará o armazenamento do referido complemento nulo como solução candidata ao ótimo. Do cumprimento ou não da condição (ii), o próximo passo que o algoritmo executa, é a poda da atual pseudo-solução parcial e o não cumprimento da condição (i), faz com que o algoritmo tente estabelecer uma pseudo-solução parcial descendente da atual.

2.5 - O ALGORITMO.

As restrições do nosso problema, têm a forma

$$b_i + \sum_{j \in N} a_{ij} x_j \geqslant 0, \qquad i \in M. \tag{II.9}$$

Chamando de y_i, $\forall\, i \in M$, a variável de folga, as restrições assumem o aspecto

$$b_i + \sum_{j \in N} a_{ij} x_j - y_i = 0, \qquad i \in M, \tag{II.10}$$

de onde tiramos naturalmente que

$$y_i = b_i + \sum_{j \in N} a_{ij} x_j, \qquad i \in M. \tag{II.11}$$

Uma pseudo-solução parcial W será viável, quando

$$y_i = b_i + \sum_{j \in W} a_{ij} x_j \geq 0, \qquad i \in M \tag{II.12}$$

e esta viabilidade se transfere automaticamente para o complemento nulo de W. Quando W não é viável, o algoritmo cria o conjunto:

$$V = \left\{ j \text{ livres} \mid z^W + c_j < z_{min},\ a_{ij} > 0 \text{ para algum } y_i < 0 \right\} \tag{II.13}$$

onde z^W é o valor de $\sum_{j \in W} c_j x_j$. Se V é vazio, W é podada. Certamente um $j \notin V$, não melhorará a cota superior z_{min} sobre a solução ótima ou não cooperará para diminuir a inviabilidade. Isto quer dizer que, se para algum $y_i < 0$, tivermos

$$y_i + \sum_{j \in V} \max \left\{ a_{ij}, 0 \right\} < 0, \tag{II.14}$$

não temos como eliminar a inviabilidade da atual pseudo-solução parcial, o algoritmo não terá como criar uma descendente para ela. V não sendo vazio ou não acontecendo (II.14), o algoritmo elege um índice $j \in V$, que reduz ao mínimo a inviabilidade da atual pseudo-solução parcial. Em outras palavras, escolhe $j_o \in V$, tal que

$$v(j_o) = \max \left\{ v(j) \mid v(j) = \sum_{i \in M} \min \left\{ y_i + a_{ij},\ 0 \right\},\ j \in V \right\}. \tag{II.15}$$

Feito isto, ele acaba de criar uma nova pseudo-solução parcial, descendente da anterior. Os testes são novamente acionados e o processo enumerativo continua dentro desta linha. Um exemplo ilustrará bem o que dissemos.

Exemplo: BALAS[2].

Seja o problema de Programação Inteira

$$\text{minimizar } z=5x_1+7x_2+10x_3+3x_4+x_5$$

sujeito a

$$\begin{aligned} -2+x_1-3x_2+5x_3+x_4+4x_5 &\geqslant 0 \\ -2x_1+6x_2-3x_3-2x_4+2x_5 &\geqslant 0 \\ -1-x_2+2x_3-x_4-x_5 &\geqslant 0 \end{aligned}$$

$$x_j \in \{0,1\}, \quad j=1,2,3,4,5$$

Iniciamos com

$$z_{min} \leftarrow \infty$$

$$W^o \leftarrow \phi, \quad x_1=\ldots=x_5=0.$$

O vetor das folgas fica

$$y=(-2,0,-1).$$

Como existem elementos negativos (basta que exista um), passamos a

$$V=\{1,3,4\}$$

Como $V \neq \phi$, verificamos se é possível eliminar a inviabilidade:

$$\begin{aligned} i=1 \qquad & -2+1+0+5+1+0 = 5 \geqslant 0 \\ i=3 \qquad & -1+0+0+2+0+0 = 1 \geqslant 0. \end{aligned}$$

Passamos à criação do descendente de W^o:

$j=1 \qquad -1-2-1 = -4$

$j=3 \qquad 0-3-0 = -3 \qquad j_o = 3 \qquad W^1 = \{ 3 \}$

$j=4 \qquad -1-2-2 = -5$

$y=(3,-3,1).$

Como $y(2) < 0$, temos

$V=\{ 2,5 \}$

Como $V \neq \phi$, tentamos vencer a inviabilidade:

$i=2 \qquad -3+0+6+0+0+2 = 5 \geqslant 0$

Descendente de W^1:

$j=2 \qquad 0+0+0 = 0 \qquad j_o = 2 \qquad W^2 = \{ 3,2 \}$

$j=5 \qquad -1-1+0 = -2$

$y=(0,3,0).$

Como $y(i) \geq 0$, $\forall i \in M$, a pseudo-solução parcial W^2 é viável e seu melhor complemento viável, é o complemento nulo.

$$c_j x_j = 10x1+7x1=17$$

e como $17 < z_{min}$, temos que

$z_{min} \leftarrow 17$ e

x "candidata" $\leftarrow (0,1,1,0,0).$

O algoritmo acaba de sondar a pseudo-solução parcial W^2, no próximo passo ela será podada, i.e., seu melhor complemento viável será enumerado e os outros complementos, implicitamente enumerados e teremos:

$W^3 = \{ 3,-2 \}$,

$$y=(3,-3,1)$$

Como $y(2) < 0$, temos

$$V=\{\ 5\ \}$$

e como $V \neq \phi$, passamos a

$$i=2 \qquad -3+2 = -1 < 0.$$

Com o índice em V não se consegue vencer a inviabilidade da restrição 2. Assim, o algoritmo poda W^3.

$$W^4=\{\ -\underline{3}\ \}$$
$$y=(-2,0,-1)$$
$$V=\{1,4\ \}$$

$$i=1 \qquad -2+1+1 = 0$$
$$i=3 \qquad -1+0+0 = -1 < 0.$$

Como a inviabilidade da terceira restrição não pode ser superada, o algoritmo poda W^4 e encerra o processo enumerativo. O algoritmo termina o processo enumerativo, quando ele poda uma pseudo-solução parcial em que todos os elementos já foram logicamente complementados. Esta é sua regra de parada. A solução ótima é a última solução que foi guardada como solução candidata e é portanto:

$$x \text{ "ótima"} = (0,1,1,0,0)$$

$$z \text{ "ótimo"} = 17.$$

A árvore gerada pelas pseudo-soluções parciais enumeradas está na figura II.2. Ela tem 5 nós e os outros implicitamente enumerados, são em número de

$$2^5 - 5 = 27.$$

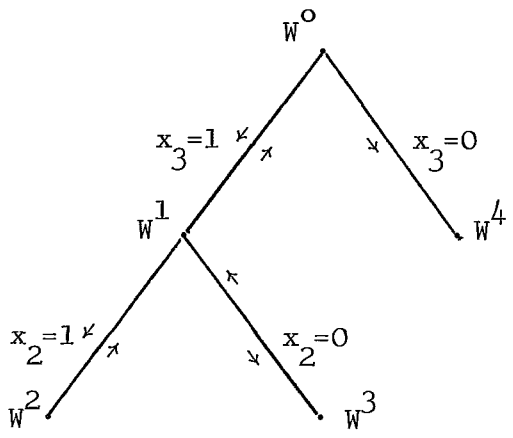


Figura II.2

Árvore das pseudo-soluções parciais do exemplo apresentado.

Colecionamos agora os passos do algoritmo.

Passo 1: Faça p=0 e $W^p=\phi$.

Passo 2: Se W^p não pode ser sondada, vá para o passo 3, senão vá para o passo 4.

Passo 3: Se existe um índice j, de variável livre, que produz uma cota superior melhor (menor) sobre a solução ótima e diminui a quantidade total de inviabilidade, aumente W^p acrescentando j à sua direita, faça p $\leftarrow$ p+1 e volte ao passo 2. Caso contrário, vá para o passo 5.

Passo 4: Se o melhor complemento viável de W^p foi encontrado e se ele produz uma cota superior melhor do que a cota produzida pela última solução candidata, armazene este complemento viável como a atual solução candidata.

Passo 5: Encontre o elemento de W^p, mais à sua direita ainda não logicamente complementado. Se não existe nenhum, termine. Caso contrário, troque-o pelo seu complemento lógico sublinhado e abandone todos os elementos imediatamente à sua direita. Faça p$\leftarrow$ p+1 e volte ao passo 2.

Teorema: GLOVER[25] e GEOFFRION[22]. A sequência de pseudo-soluções parciais gerada pelo algoritmo acima, é não redundante e termina somente quando todas as 2^n pseudo-soluções parciais forem explícita e/ou implicitamente enumeradas.

Os principais passos de uma demonstração por indução, GEOFFRION[22], baseada na forma de podagem das pseudo-soluções parciais pelo algoritmo, são:

Se o algoritmo pode podar W^o, as condições do teorema ficam satisfeitas. Caso contrário, tomando por hipótese (de indução) que $W^1,\ldots,W^p$, $p \geqslant 1$, são não redundantes, a prova de que W^{p+1} é não redundante, garantirá a não redundância de toda a sequência. W^{p+1} é não redundante ou porque é uma descendente de W^p e nesse caso $W^p \subset W^{p+1}$ ou porque W^{p+1} contém pelo menos um elemento logicamente complementar de algum elemento em cada pseudo-solução parcial previamente podada. Estes fatos são vistos melhor se desmembrarmos os passos do algoritmo, responsáveis pela passagem de W^p para W^{p+1}. Assim, W^{p+1} é obtida de W^p:

i. pelo aumento à direita de W^p com um índice de variável livre. (Aqui W^{p+1} é uma descendente de W^p).

ii. pela troca do último elemento de W^p pelo seu complemento lógico sublinhado.

iii. pela troca do último elemento de W^p, ainda não logicamente complementado, pelo seu complemento lógico sublinhado e abandono de todos os elementos à direita deste último logicamente complementado.

As 2^n pseudo-soluções parciais são implicitamente enumeradas, somente depois que a pseudo-solução parcial onde todos os elementos foram logicamente complementados, for implícita ou explicitamente enumerada. A sua enumeração é garantida pelas regras de fixar e liberar as variáveis.

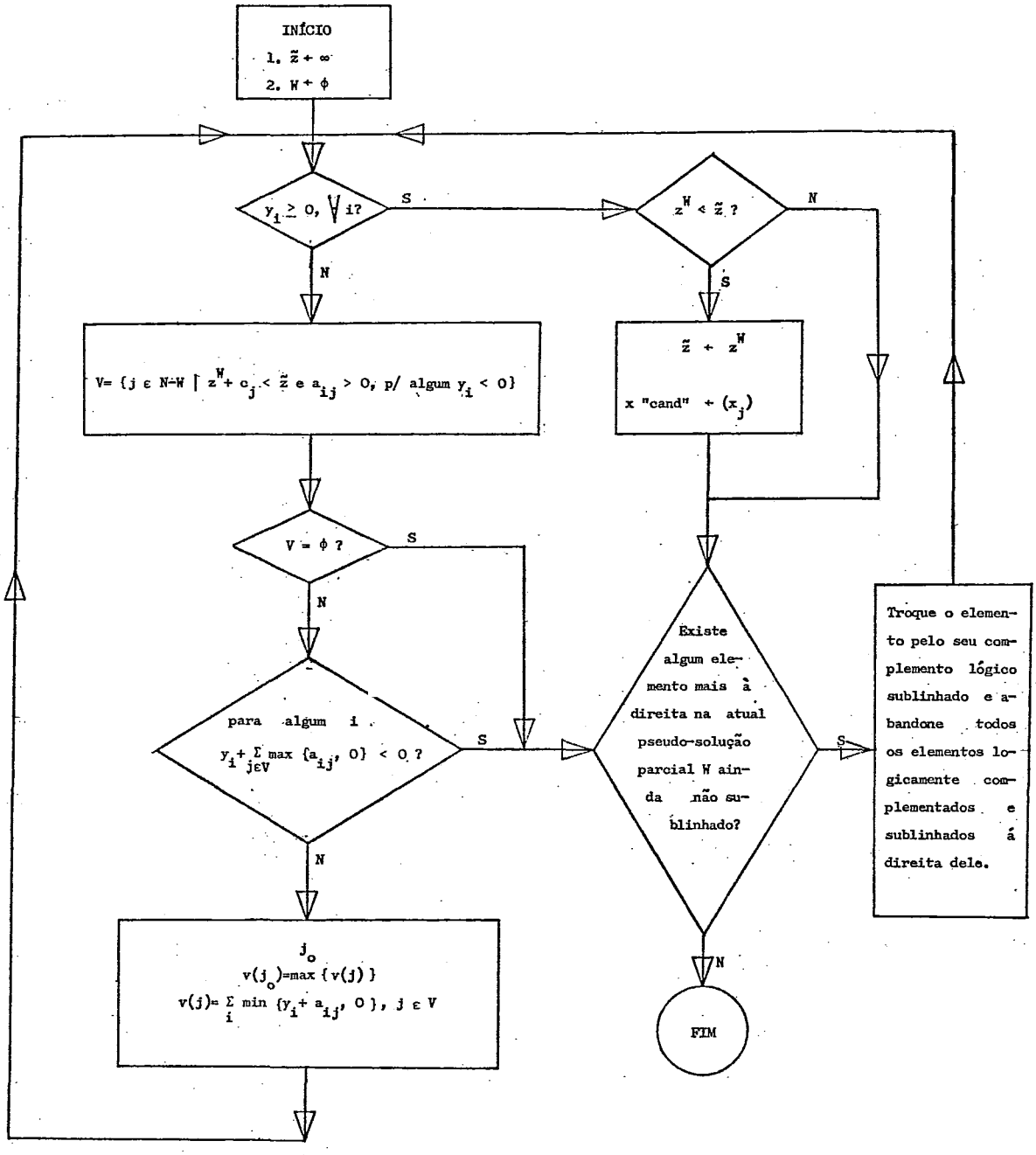


Figura II.3

Fluxograma para o algoritmo de enumeração implícita das pseudo-soluções parciais de um problema de Programação Inteira bivalente.

2.6 - ACELERAÇÃO DO ALGORITMO, FILTROS OU RESTRIÇÕES SUBSTITUTAS.

Vários projetos de aceleração do algoritmo de BALAS[2], foram levados a efeito e alguns com muito sucesso. Os principais e melhores avanços, na nossa opnião, foram produzidos por GLOVER[26], PETERSEN[46] e GEOFFRION[23]. O segundo, dentre outras coisas, estabeleceu como gerar uma pseudo-solução parcial viável inicial, i.e., uma origem para iniciar o processo de enumeração. Trata-se de começar com uma pseudo-solução parcial viável, onde os índices das variáveis são obtidos da ordenação crescente dos valores de

$$c_j / \sum_{i \in M} a_{ij}, \qquad a_{ij} > 0, \qquad j \in N. \tag{II.16}$$

De fato, os valores mínimos da razão são produzidos pelos menores custos c_j, $j \in N$ e pelas maiores somas

$$\sum_{i \in M} a_{ij}, \qquad a_{ij} > 0, \qquad j \in N \tag{II.17}$$

e portanto, pelos índices que têm grande possibilidade de participar da solução ótima (menores custos porque estamos minimizando e maiores somas (II.17), porque estamos tentando vencer inviabilidades).

GLOVER[25,26] mostrou que melhores informações sobre viabilidade e otimalidade relativas às variáveis livres em problemas de Programação Inteira bivalente, podem ser obtidas pela combinação criteriosa das restrições originais do problema, numa única restrição, chamada de "Filtro" ou "restrição substituta". Usaremos a abreviação "s-restrição" para esses filtros. GEOFFRION[23], mostrou como gerar essas s-restrições através de soluções produzidas por variáveis duais relativas às restrições de subproblemas do problema proposto. Estes subproblemas, como veremos, são construidos a partir das

variáveis livres relativas a uma pseudo-solução parcial. A definição que segue formaliza melhor a idéia de s-restrição.

Uma s-restrição é uma combinação linear não negativa das restrições do problema original.

Se $b+Ax \geqslant 0$, descreve o conjunto das restrições de um problema de Programação Inteira bivalente, uma s-restrição, definida a partir delas, tem a forma

$$u(b+Ax) \geqslant 0, \qquad \text{(II.18)}$$

onde $u=(u_1,\ldots,u_m)$ é tal que $u_i \geqslant 0$, $\forall i \in M$.

Para ilustrar a eficácia das s-restrições, tomemos um simples problema de Programação Inteira bivalente em que as duas únicas restrições são:

$$1.\ x_1 - 2x_2 \geqslant 1 \quad \text{e} \quad 2.\ -2x_1 + x_2 \geqslant 1.$$

Uma solução viável para a primeira, é (1,0) e uma viável para a segunda é (0,1). Agora, se construirmos uma s-restrição com u=(1,1), teremos

$$-x_1 - x_2 \geqslant 2,$$

que mostra a inviabilidade binária do problema.

Sobre as s-restrições, temos dois fatos importantes:

a. Se x^o é uma solução viável do problema original ($b+Ax^o \geqslant 0$), x^o é também viável para qualquer s-restrição deste problema ($u(b+Ax^o) \geqslant 0$).

b. Se uma s-restrição não tem solução viável, o problema original, também não tem.

O fato (a) que dizer também que uma s-restrição é uma restrição redundante

no sentido usual. O fato (b), sugere que a criação de testes que o considerem, pode apressar o processo enumerativo.

GEOFFRION[23] propos uma s-restrição compondo aquela de GLOVER[26] com a restrição $z_{min} - cx \geqslant 0$, querendo com isso que uma solução fosse viável para a s-restrição resultante quando também produzisse um valor para a função objetivo melhor do que a atual cota superior guardada em z_{min}. Ou seja, a s-restrição proposta, tem a forma

$$u(b+Ax)+(z_{min} - cx) \geqslant 0. \qquad \text{(II.19)}$$

Consideremos agora uma restrição da forma

$$b_k + \sum_j a_{kj} x_j \geqslant 0. \qquad \text{(II.20)}$$

Dizemos que ela é binário inviável, se ela não tem solução binária, i.e., se não existe um vetor binário que a satisfaça e é binário inviável condicionalmente, se sua viabilidade binária é restrita a determinados valores das variáveis x_j. Uma outra forma de expressarmos isto, é

1) $b_k + \sum_j a_{kj} x_j \geqslant 0$ (> 0) é binário inviável se, e somente se,

$$b_k + \sum_j \max\left\{a_{kj}, 0\right\} < 0 \ (\leqslant 0).$$

2) Se $b_k + \sum_j \max\left\{a_{kj}, 0\right\} - |a_{kj^*}| < 0 \ (\leqslant 0)$, então

$x_{j^*} = 0$ ou $x_{j^*} = 1$, conforme $a_{kj^*} < 0$ ou $a_{kj^*} > 0$, em qualquer solução binária de

$$b_k + \sum_j a_{kj} x_j \geqslant 0 \ (> 0).$$

Um conceito de "poder" introduzido para as s-restrições, no sentido de sua eficiência quando submetidas a testes de viabilidade binária, foi também proposto por GEOFFRION[23].

Relativamente a uma pseudo-solução parcial W, a s-restrição

$$u_1(b+Ax) + (z_{min} - cx) \geqslant 0, \qquad u_1 \geqslant 0,$$

é mais forte do que a s-restrição

$$u_2(b+Ax) + (z_{min} - cx) \geqslant 0, \qquad u_2 \geqslant 0,$$

se

$$\max_x \left\{ u_1(b+Ax) + (z_{min} - cx) \right\} < \max_x \left\{ u_2(b+Ax) + (z_{min} - cx) \right\},$$

onde os máximos são tomados sobre as variáveis livres, i.e., sobre as variáveis x_j, tais que $j \notin W$.

GLOVER[26] usou uma definição de "poder" semelhante a esta:

Relativamente a uma pseudo-solução parcial W, a s-restrição

$$u_1(b+Ax) \geqslant 0, \qquad u_1 \geqslant 0,$$

é mais forte do que

$$u_2(b+Ax) \geqslant 0, \qquad u_2 \geqslant 0,$$

se o máximo de $z_{min} - cx$, sujeito a $u_1(b+Ax) \geqslant 0$, é menor do que o máximo de $z_{min} - cx$, sujeito a $u_2(b+Ax) \geqslant 0$, os máximos sendo tomados sobre as variáveis livres.

O problema de determinar s-restrições poderosas no sentido das definições acima, é o de minimizar sobre todos os $u \geqslant 0$, as expressões

$$\max_x \left\{ u(b+Ax) + (z_{min} - cx) \right\},$$

ou seja

$$\min_{u}\left\{\max_{x}\left\{u(b+Ax)+(z_{min}-cx),\ x\ \text{livre}\right\},\ u\geqslant 0\right\}.$$

Nosso próximo passo, é mostrar como os $u \geqslant 0$ podem ser obtidos como variáveis duais relativas às restrições do problema original. Este sendo transformado num subproblema pela substituição dos valores das variáveis fixadas pela pseudo-solução parcial, o que não altera o número das restrições originais.

Tomemos então o problema

$$\min_{u\geqslant 0}\ \max_{x}\left\{u(b+Ax)+(z_{min}-cx),\ x\ \text{livre}\right\}.$$

Desenvolvendo sua parte de maximização, obtemos

$$\max_{x}\Big\{\sum_{i\in M}u_i\Big(b_i+\sum_{j\in W}a_{ij}x_j^W+\sum_{j\notin W}a_{ij}x_j\Big)+\Big(z_{min}-\sum_{j\in W}c_jx_j^W-\sum_{j\notin W}c_jx_j\Big)\ \Big|\ u_i\geqslant 0,$$

$$\forall i\in M,\ x_j\in\{0,1\},\ j\notin W\Big\}. \qquad \text{(II.21)}$$

Fazendo

$$y_i^W=b_i+\sum_{j\in W}a_{ij}x_j^W \qquad \text{e}$$

$$z^W=\sum_{j\in W}c_jx_j^W$$

e substituindo em (II.21), vem

$$\sum_{i\in M}y_i^Wu_i+z_{min}-z^W+\max_{x}\Big\{\sum_{j\notin W}(a_{ij}u_i-c_j)x_j\ \Big|\ x_j\in\{0,1\},\ j\notin W\Big\}. \qquad \text{(II.22)}$$

Trocando as restrições $x_j \in \{0,1\}$ por $0 \leqslant x_j \leqslant 1$, não afetamos as soluções do problema, dado que no caso contínuo o máximo é assumido nos extremos. Assim, teremos para o lugar de (II.22) o seguinte

$$\sum_{i \in M} y_i^W u_i + z_{min} - z^W + \max_x \left\{ \sum_{j \notin W} (a_{ij} u_i - c_j) x_j \mid 0 \leqslant x_j \leqslant 1,\ j \notin W \right\}. \qquad \text{(II.23)}$$

O dual da parte

$$\max_x \left\{ \sum_{j \notin W} (a_{ij} u_i - c_j) x_j \mid 0 \leqslant x_j \leqslant 1,\ j \notin W \right\}, \qquad \text{(II.24)}$$

é

$$\min \left\{ \sum_{j \notin W} v_j \mid v_j \geqslant \sum_{i \in M} a_{ij} u_i - c_j \text{ e } v_j \geqslant 0,\ j \notin W \right\}. \qquad \text{(II.25)}$$

Finalmente temos a forma do problema de programação linear que produzirá os coeficientes $u_i \geqslant 0$, para a geração dos filtros ou restrições substitutas, dada por

$$(PPL_W) \qquad \text{minimizar } t = \sum_{i \in M} y_i^W u_i + z_{min} - z^W + \sum_{j \notin W} v_j, \qquad \text{(II.26)}$$

$$\text{sujeito a} \quad \sum_{i \in M} a_{ij} u_i - v_j \leqslant c_j \qquad \text{(II.27)}$$

$$u_i \geqslant 0, \quad v_j \geqslant 0, \quad i \in M, \quad j \notin W. \qquad \text{(II.28)}$$

Vejamos algumas informações interessantes produzidas pelas

soluções de (PPL_W). Chamando de t_{min} a solução mínima (ótima) de (PPL_W), temos o seguinte:

i. Pela parte (1) dos testes de viabilidade binária, a atual pseudo-solução parcial W, não tem remate viável se $t_{min} < 0$. O cálculo de t_{min} pode ser interrompido no momento em que este valor se tornar negativo. Neste caso, o algoritmo poda a pseudo-solução parcial W.

ii. Por outro lado, se $t_{min} \geqslant 0$, o algoritmo não poda a atual pseudo-solução parcial W e os valores dos u_i's são utilizados para gerar a melhor restrição substituta relativa à pseudo-solução parcial W e a parte (2) dos testes de viabilidade binária, fixa convenientemente variáveis livres nos níveis zero ou um, criando a pseudo-solução parcial descendente de W, agora possivelmente, pelo acréscimo de vários índices de variáveis livres na atual pseudo-solução parcial W.

A s-restrição introduzida por GEOFFRION[23], pode ser fortalecida se obrigarmos a uma pseudo-solução parcial, satisfazer

$$cx \leqslant z_{min} - 1, \tag{II.29}$$

no lugar de

$$cx \leqslant z_{min} \tag{II.30}$$

como mais uma condição para que ela seja viável. Neste caso, a s-restrição ficaria:

$$u(b+Ax) + (z_{min} - 1 - cx) \geqslant 0. \tag{II.31}$$

Vejamos agora um exemplo para fortalecer a compreensão dos fatos abordados.

Exemplo:

$$\text{minimizar } z = 3x_1 + 2x_2 + 5x_3 + 2x_4 + 3x_5$$

sujeito a

$$\begin{aligned} 1 + x_1 + x_2 - x_3 - 2x_4 + x_5 &\geq 0 \\ -2 + 7x_1 - 3x_3 + 4x_4 + 3x_5 &\geq 0 \\ -1 - 11x_1 + 6x_2 + 3x_4 + 3x_5 &\geq 0 \end{aligned}$$

$$x_j \in \{0,1\}, \quad j=1,2,3,4,5.$$

Aplicando primeiro os testes do algoritmo de enumeração implícita para o cálculo de uma pseudo-solução parcial viável inicial, obtemos

$$W^1 = \{5\}, \qquad y^{W^1} = (2,1,2) \quad \text{e}$$

$$z_{min} = 3.$$

O algoritmo poda a pseudo-solução parcial W^1 e o resultado, é

$$W^2 = \{-\underline{5}\}, \qquad y^{W^2} = (1,-2,-1) \quad \text{e} \quad z^{W^2} = 0.$$

Agora, o PPL_W que determinará os coeficientes u_i's com os quais a s-restrição será construida, é

$$\text{minimizar } t^{W^2} = u_1 - 2u_2 - u_3 + v_1 + v_2 + v_3 + v_4 + (z_{min} - 1 - z^{W^2})$$

$$\begin{aligned} \text{sujeito a } -3 + u_1 + 7u_2 - 11u_3 - v_1 &\leq 0 \\ -2 + u_1 + 6u_3 - v_2 &\leq 0 \\ -5 - u_1 - 3u_2 - v_3 &\leq 0 \\ -2 - 2u_1 + 4u_2 + 3u_3 - v_4 &\leq 0, \quad u_i \geq 0 \text{ e } v_j \geq 0. \end{aligned}$$

As soluções são:

$$u_1 = 0, \quad u_2 = 0,48 \quad \text{e} \quad u_3 = 0,03.$$

$$t_{min}^{W^2} = -0,99 + (z_{min} - 1 - z^{W^2}) =$$

$$= -0,99 + 3 - 1 - 0 = 1,01 > 0.$$

Como $t_{min}^{W^2} > 0$, W^2 não é podada e a s-restrição é construida:

$$u_1(1 + x_1 + x_2 - x_3 - 2x_4) + u_2(-2 + 7x_1 - 3x_2 + 4x_4) + u_3(-1 - 11x_1 + 6x_2 + 3x_4) \geqslant 0.$$

Depois de serem feitas as contas, vem

$$1,01 + 0,03x_1 + 1,82x_2 - 6,44x_3 + 0,01x_4 \geqslant 0.$$

Aplicando nesta s-restrição os testes de viabilidade binária (mais especificamente, a parte (2) dos referidos testes), obtemos que as variáveis livres x_2 e x_3, devem ser fixadas no nível zero na próxima pseudo-solução parcial, para que esta tenha pelo menos um remate viável para a s-restrição. Informações adicionais como estas, não são providas pelas restrições originais do problema. Lembrando que uma pseudo-solução parcial viável para o conjunto original de restrições é também viável para a s-restrição, mas que, entretanto, a afirmação contrária não é necessariamente verdadeira, criada uma pseudo-solução parcial viável para a s-restrição, temos necessidade de testar sua viabilidade relativamente ao conjunto original de restrições. Caso exista inviabilidade, os testes do algoritmo de enumeração implícita se encarregam de criar uma pseudo-solução parcial descendente desta, quando isto é possível e quando não, o algoritmo poda, como de costume, a presente pseudo-solução parcial. Agora, a pseudo-solução parcial descendente de W^2, é

$$W^3 = \{ -\underline{5}, -2, -3 \}$$

Como ela não tem complemento viável para o conjunto original de restrições, os testes do algoritmo elegem entre as variáveis livres, x_1 e x_4, a última variável, formando assim, a pseudo-solução parcial W^4, descendente de W^3, ou seja

$$W^4 = \{ -5, -2, -3, 4 \}.$$

O próximo PPL_W tem uma única restrição relativamente à única variável livre x_1. No novo PPL_W, t^W torna-se negativo, o que mostra a impossibilidade de ser construida uma pseudo-solução parcial com complemento viável para a s-restrição que seria construida e portanto para o conjunto original de restrições. O algoritmo poda a atual pseudo-solução parcial e prossegue assim, até a obtenção de informações finais. Neste exemplo, ele não consegue exibir uma pseudo-solução parcial que forneça uma cota superior melhor do que $z_{min} = 3$ e esta fica sendo portanto, o valor da função objetivo relativo a uma solução ótima encontrada, que é (0,0,0,0,1).

2.7 - ALGUMAS OBSERVAÇÕES ADICIONAIS.

Finalizando este capítulo, faremos algumas observações.

No algoritmo original de enumeração implícita a passagem de uma pseudo-solução parcial W^p para a sua descendente W^{p+1}, era feita pelo acréscimo de um único índice de variável livre em W^p. Isto pode, evidentemente, ser feito pelo acréscimo de vários índices simultaneamente.

As soluções ótimas alternativas podem ser obtidas construindo W^o com uma permutação de uma solução ótima, fazendo z_{min} igual ao valor ótimo da função objetivo e trocando a restrição

$$cx + c_j < z_{min}$$

da construção do conjunto V pela restrição

$$cx + c_j \leqslant z_{min}.$$

Neste caso, cada solução candidata ao ótimo, é uma solução ótima. Se para al

gum j_o, tivermos $c_{j_o} = 0$, ao fim da enumeração das pseudo-soluções parciais, não teremos encontrado todas as soluções ótimas necessariamente, pois se x_{j_o} na solução ótima ficou no nível zero, $x_{j_o} = 1$, se não alterar a viabilidade, estará numa solução ótima também.

O algoritmo acrescenta ou libera índices de variáveis nos sucessivos W's, obedecendo a disciplina: "último a chegar, primeiro a sair". Esta não é a única nem necessariamente a melhor regra.

CAPÍTULO III

OS MODELOS DE RECOBRIMENTO E PARTICIONAMENTO.

3.1 - INTRODUÇÃO.

Trataremos aqui com uma classe de problemas dentro da otimização combinatória, também do tipo 0-1, cuja modelagem para estudo de muitas situações reais, é feita com a idéia do que chamamos de "conjunto recobrimento" e "conjunto particionamento".

3.2 - CONJUNTO RECOBRIMENTO E CONJUNTO PARTICIONAMENTO.

Sejam $M = \{1,\ldots,m\}$, $N = \{1,\ldots,n\}$ e $K = \{K_1,\ldots,K_n\}$, três conjuntos tais que para cada $j \in N$, $K_j \subset M$.

Definição 1:

Chamamos de "conjunto recobrimento" de M a um subconjunto $N^* \subset N$, tal que $\bigcup_{j \in N^*} K_j = M$.

Definição 2:

Chamamos de "conjunto particionamento" de M a um subconjunto $N^* \subset N$, tal que

i. $\bigcup_{j \in N^*} K_j = M$ e

ii. $K_s \cap K_t = \phi$, para dois índices s e t quaisquer, tais que $s \in N^*$, $t \in N^*$ e $s \neq t$.

3.3 - DEFINIÇÃO DOS MODELOS.

Se para cada $j \in N$, chamarmos de c_j ($c_j > 0$) o custo associado a este índice, teremos que o custo total do conjunto recobrimento $N^* \subset N$, será dado por

$$\sum_{j \in N^*} c_j . \qquad \text{(III.1)}$$

Propomos o problema de encontrar o conjunto recobrimento de custo mínimo. Para criar o modelo de recobrimento, definimos uma variável $x_j \in \{0,1\}$ para cada $j \in N$, tal que, na solução do problema, $x_j = 1$ se j pertence ao conjunto recobrimento de custo mínimo e $x_j = 0$, caso contrário. A matriz $A=(a_{ij})$ do modelo, é tal que $a_{ij} = 1$ quando o elemento $j \in N$ recobre o elemento $i \in M$, (i.e., $i \in K_j \subset M$) e $a_{ij} = 0$, caso contrário. Chamaremos este, problema do conjunto recobrimento (PCR) e a forma de seu modelo será a de um problema de Programação Inteira bivalente:

(PCR) minimizar $z = \sum_{j \in N} c_j x_j$, $N=\{1,\ldots,n\}$ (III.2)

sujeito a $\sum_{j \in N} a_{ij} x_j \geq 1$, $i \in M=\{1,\ldots,m\}$ (III.3)

$x_j \in \{0,1\}$, $j \in N$ (III.4)

$a_{ij} \in \{0,1\}$, $i \in M$, $j \in N$. (III.5)

Também podemos procurar a solução do problema do conjunto particionamento de custo mínimo (PCP), bastando para tanto trocar (III.3) no modelo do (PCR), por

$$\sum_{j \in N} a_{ij} x_j = 1, \qquad i \in M=\{1,\ldots,m\}. \tag{III.6}$$

No (PCR) procuramos a união mais barata dos conjuntos K_j's, tais que cada $i \in M$, pertence a "pelo menos um" K_j (i.e., cada $i \in M$ é recoberto por "pelo menos um" K_j). No (PCP) também procuramos a união de menor custo dos conjuntos K_j's, mas agora com a condição de que cada $i \in M$, pertença a "um único" K_j (cada $i \in M$ é recoberto por "um único" K_j). Para tanto, os K_j's devem ser disjuntos. São muitas as situações reais, basicamente modeláveis dessa forma, dado que é muito frequente a utilização de uma matriz $A=(a_{ij})$ mxn, cujas entradas são inteiros 0-1, para representar a distribuição de n elementos para m conjuntos: 1's na linha $i \in M$, designam os elementos que ocorrem no i-ésimo conjunto e 1's na coluna $j \in N$, designam os conjuntos que contém o j-ésimo elemento. Tais matrizes são consideradas fundamentais em investigações combinatórias.

Para ilustrar, vejamos o seguinte: imaginemos a disponibilidade de n recursos para a execução de m tarefas. Cada recurso $j \in N=\{1,\ldots,n\}$ pode ser alocado para a execução de algumas tarefas (ou todas). Com isso, podemos construir uma matriz (catálogo) de todas as formas possíveis de associar um ou mais recursos a uma dada tarefa, sujeito a uma série de restrições como localização, demanda, espaço, tempo, condições geográficas, climáticas, etc. Assim, na matriz catálogo, as linhas correspondem às tarefas e as colunas, às possíveis combinações das tarefas para os recursos (i.e., as formas possíveis de cada recurso j ser utilizado). Assim, uma entrada $a_{ij(k)}$ da matriz é 1(0), se a k-ésima combinação o recurso j é (não é) utilizada pela tarefa i. Atribuindo um custo $c_{j(k)}$ à utilização da k-ésima combinação do recurso j (i.e., k-ésima combinação que utiliza o recurso j), podemos estabelecer um modelo para o problema, definindo uma variável $x_{j(k)} \in \{0,1\}$, que será igual a 1 na solução, se a k-ésima combinação do recurso j é utilizada e 0 (zero), caso contrário. A garantia do cumprimento de cada tarefa, "pelo menos uma vez", ou utilizando "pelo menos um recurso", é dada por

$$\sum_{k}\sum_{j} a_{ij(k)}x_{j(k)} \geqslant 1, \qquad \text{(III.7)}$$

para cada tarefa $i \in M=\{1,\ldots,m\}$ e se quizermos garantir o cumprimento de cada tarefa "uma única vez", ou utilizando "um único" recurso, obrigamos que

$$\sum_{k}\sum_{j} a_{ij(k)}x_{j(k)} = 1, \qquad i \in M, \qquad \text{(III.8)}$$

onde $a_{ij(k)} \in \{0,1\}$. Como desejamos minimizar o custo total de execução das tarefas, temos finalmente o modelo completo dado por

$$\text{minimizar} \quad z = \sum_{k}\sum_{j} c_{j(k)}x_{j(k)} \qquad \text{(III.9)}$$

$$\text{sujeito a} \quad \sum_{k}\sum_{j} a_{ij(k)}x_{j(k)} \geqslant 1, \quad i \in M \qquad \text{(III.10)}$$

$$x_{j(k)} \in \{0,1\}, \qquad a_{ij(k)} \in \{0,1\}. \qquad \text{(III.11)}$$

O quadro da figura III.1, dá uma ilustração do que dissemos. Observamos na definição dos modelos dos (PCR) e (PCP) que as restrições $x_j \in \{0,1\}$, $j \in N$, podem ser substituidas por $x_j \geqslant 0$ e inteiro, $\forall\, j \in N$. Uma equivalência entre as duas formas, é dada pelo fato de que em qualquer solução mínima, a condição $x_j \geqslant 0$ e inteiro, $\forall\, j \in N$, implica $x_j \in \{0,1\}$, (a implicação contrária é óbvia). De fato, se numa solução mínima (ótima), as restrições

$$\sum_{j \in N} a_{ij}x_j \geqslant 1 \qquad \text{(III.12)}$$

Variáveis	$x_{1(1)}$	$x_{1(2)}$	$x_{1(3)}$ ...	$x_{2(1)}$	$x_{2(2)}$ ...
Custos	$c_{1(1)}$	$c_{1(2)}$	$c_{1(3)}$ ...	$c_{2(1)}$	$c_{2(2)}$ ...
Recursos		1			2
Combinações	1	2	3	1	2
Tarefas					
a^1	1	0	0	0	1
a^2	1	0	1	1	1
a^3	0	1	0	1	0
a^4	1	1	0	0	0
a^5	1	1	0	1	0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
a^m	0	0	1	0	1

Figura III.1

são satisfeitas e pelo menos um x_j é maior que 1 ($x_j > 1$), a redução desta componente a 1, continua satisfazendo às restrições (III.12) e como os custos são positivos, temos neste caso uma solução melhor do que a anterior, o que contradiz o fato da solução com uma componente $x_j > 1$, ser ótima.

3.4 - CONVERSÃO DE UM (PCP) EM UM (PCR).

Naturalmente, o (PCR) é mais geral do que o (PCP), posto que qualquer solução viável do (PCP) é também viável para o (PCR) e pela modificação dos custos, podemos transformar um (PCP) num (PCR), desde que o (PCP) tenha uma solução viável. Com isso, utilizando um algoritmo que busque as soluções do (PCR), podemos encontrar as soluções do (PCP), quando elas existirem. Chamando de y_i, $i \in M$, as variáveis de folga do (PCR), i.e.,

$$y_i = \sum_{j \in N} a_{ij} x_j - 1, \qquad i \in M, \tag{III.13}$$

temos que o (PCP) estará resolvido se a solução do correspondente (PCR) (solução ótima, mínima), for obtida com todas as variáveis de folga iguais a zero. Atribuindo a cada variável de folga um custo suficientemente alto, por exemplo, um custo maior do que a soma de todos os custos, temos a possibilidade de obtermos a solução mínima do (PCR) com todas as variáveis de folga nulas e esta solução mínima é também solução mínima do (PCP). Caso numa solução como esta, as variáveis de folga não sejam todas nulas, temos a evidência de que o (PCP) não tem solução.

O (PCP) correspondente ao (PCR)

$$\text{minimizar} \quad z = \sum_{j \in N} c_j x_j \tag{III.14}$$

$$\text{sujeito a} \quad \sum_{j \in N} a_{ij} x_j \geqslant 1, \qquad i \in M \tag{III.15}$$

$$a_{ij} \in \{0,1\}, \quad x_j \in \{0,1\}, \quad i \in M, \quad j \in N, \tag{III.16}$$

é equivalente a

$$\text{minimizar} \quad z = \sum_{j \in N} c_j x_j + V \cdot \sum_{i \in M} y_i \qquad \text{(III.17)}$$

$$\text{sujeito a} \quad \sum_{j \in N} a_{ij} x_j - y_i = 1, \quad i \in M, \qquad \text{(III.18)}$$

$$a_{ij} \in \{0,1\}, \quad x_j \in \{0,1\}, \quad y_i \geq 0, \; i \in M, \; j \in N \qquad \text{(III.19)}$$

onde $V > \sum_{j \in N} c_j$. Vejamos o comportamento das soluções dos (PCP) e (PCR), após ligeira transformação dos custos do (PCP), formalizado no

Teorema: Sejam $A=(a_{ij})$ uma matriz mxn, $c=(c_1,\ldots,c_n)$ o vetor dos custos e

$$p_j = \sum_{i=1}^{m} a_{ij}, \qquad j \in N.$$

Escolhendo qualquer número $V > \Sigma\, c_j$, $j \in N$, temos que, se o (PCP) definido com a matriz A e o vetor dos custos c, tem uma solução ótima, então o (PCR) definido por A e pelos custos $c'_j = c_j + V.p_j$, para todo $j \in N$, tem a mesma solução ótima.

Demonstração:

Chamando de S_R e S_P respectivamente, os conjuntos das soluções dos (PCR) e (PCP), temos imediatamente que

$$S_P \subset S_R,$$

ou seja, as soluções de particionamento são também de recobrimento. Interessa-nos o fato de que

$$S_R \subset S_P.$$

Para demonstrá-lo, imaginemos que exista pelo menos uma solução de recobrimento que não seja de particionamento, i.e., imaginemos que

$$S_R - S_P \neq \phi .$$

Tomemos

$$\tilde{x} \in S_R - S_P \quad \text{e} \quad x' \in S_P.$$

Naturalmente, $x' \in S_P$ implica $x' \in S_R$. Chamando de p o vetor

$(p_j)=(p_1,\ldots,p_n)$, temos

$$c'x'=cx'+ Vpx'.$$

Observando que

$$V \geq \sum_j c_j \geqslant \sum_j c_j x'_j = cx'$$

e que

$$Vpx'=V(p_1,\ldots,p_n) \begin{bmatrix} x'_1 \\ \cdot \\ \cdot \\ \cdot \\ x'_n \end{bmatrix}$$

$$=V\left(\sum_i a_{i1},\ldots,\sum_i a_{in}\right) \begin{bmatrix} x'_1 \\ \cdot \\ \cdot \\ \cdot \\ x'_n \end{bmatrix}$$

$$=V \sum_j \sum_i a_{ij} x'_j$$

$$=V \sum_i \sum_j a_{ij} x'_j = V \cdot \sum_{i=1}^{m} 1 = V.m,$$

temos que

$$c'x' < V + V.m = V(m + 1). \qquad \text{(III.20)}$$

Agora

$$c'\tilde{x} = c\tilde{x} + Vp\tilde{x}$$

$$c'\tilde{x} = c\tilde{x} + V\left(\sum_i a_{i1}, \ldots, \sum_i a_{in}\right) \begin{bmatrix} \tilde{x}_1 \\ \cdot \\ \cdot \\ \cdot \\ \tilde{x}_n \end{bmatrix}$$

$$= c\tilde{x} + V(1, \ldots, 1) \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \cdot \\ \cdot \\ \cdot \\ \tilde{x}_n \end{bmatrix}$$

$$= c\tilde{x} + (V, \ldots, V) \begin{bmatrix} b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_m \end{bmatrix}$$

pelo menos um b_i é maior que 1. Assim, podemos decompor o vetor (b_i) no vetor $(1 + b_i')$ e pelo menos um vetor b_i' será diferente de zero. Temos

$$c'\tilde{x} = c\tilde{x} + (V, \ldots, V) \begin{bmatrix} 1 + b_1' \\ \cdot \\ \cdot \\ \cdot \\ 1 + b_m' \end{bmatrix}$$

$$= c\tilde{x} + V.m + V \cdot \sum_{i \in M} b_i' \qquad \text{(III.21)}$$

Chamando de F o conjunto

$$\left\{ i \mid \sum_{j \in N} a_{ij} \tilde{x}_j > 1 \right\}$$

e observando que $card(F) \geqslant 1$, ou seja, que existe pelo menos uma restrição que é satisfeita com a desigualdade estrita ($>$), temos

$$card(F) \leqslant \sum_{i \in M} b'_i \qquad \text{(III.22)}$$

já que estamos tratando com números inteiros e portanto, cada b'_i ou é zero ou satisfaz $b'_i \geqslant 1$ e para cada i que $b'_i \geqslant 1$ ocorra, a cardinalidade de F é aumentada de 1 (um).

Substituindo (III.22) em (III.21), obtemos

$$c'\tilde{x} \geqslant c\tilde{x} + V.m + V.card(F)$$

$$= c\tilde{x} + V(m + card(F))$$

$$\geqslant c\tilde{x} + V(m + 1)$$

$$\geqslant V(m + 1) > c'x'.$$

Ora, $x' \in S_R$ e $c'\tilde{x} > c'x'$. Como por hipótese $\tilde{x}$ é solução de recobrimento, a desigualdade traduz uma contradição para o fato de que existe uma solução de recobrimento em $S_R - S_P$, ou seja, a situação correta é descrita por

$$S_R - S_P = \phi ,$$

provando a igualdade dos conjuntos das soluções de recobrimento e particionamento e consequentemente, a equivalência dos dois problemas nessas condições.

Vejamos agora um

Exemplo:

Seja o problema

$$\text{minimizar } z = 2x_1 + 3x_2 + 6x_3 + x_4$$

$$\text{sujeito a} \quad x_1 \qquad\qquad + x_4 = 1$$

$$+ x_2 \qquad + x_4 = 1$$

$$x_1 + x_3 \qquad = 1$$

$$x_1 + x_3 \qquad = 1$$

$$x_j \in \{0,1\}, \quad j=1,2,3,4.$$

A solução ótima deste (PCP) é (1,1,0,0) e o (PCR) obtido a partir deste pela troca das restrições de igualdade pelas correspondentes restrições do tipo $\geqslant$, tem como solução ótima (1,0,0,1). Fazendo

$$V = 13 > \sum c_j = 12,$$

o vetor dos custos $c' = c + V \cdot \sum_i a_{ij}$, $j \in N$, é dado por (41,16,32,27) e a solução ótima do (PCR), é (1,1,0,0).

3.5 - REDUÇÕES (ELIMINAÇÃO DE LINHAS E COLUNAS DA MATRIZ DOS MODELOS DE RECOBRIMENTO E PARTICIONAMENTO).

Existem situações em que podemos eliminar linhas e colunas da matriz do modelo, sem que isso afete as soluções do problema e torne menos

penosa a busca das mesmas. Algumas eliminações são válidas para os dois problemas e outras para um único. Indicaremos isso claramente. Representaremos por lin(i) a i-ésima linha da matriz e por col(j) a j-ésima coluna.

1. Recobrimento e Particionamento. Se lin(i) é o vetor nulo para algum $i \in M$, nenhum dos dois problemas tem solução viável, uma vez que a i-ésima restrição não pode ser satisfeita. Uma situação como esta decorre da não consistência do problema ou dos dados ou ainda de sua má formulação.

2. Recobrimento e Particionamento. Se $lin(i)=e_q$ (e_q é o vetor unitário onde a q-ésima componente é igual a 1) para algum $i \in M$, $q \in N$, então $x_q = 1$ em toda solução viável e a col(q) pode ser suprimida. Cada linha p, tal que $p \in K_q$, i.e., $a_{pq} = 1$, pode ser suprimida, uma vez que é impossível recobrir a lin(i) sem a col(q) e cada elemento de K_q é recoberto pela col(q).

3. Particionamento. Para evitar um "super-recobrimento" da linha $s \in M$, junto com a supressão da redução (2), cada coluna $k \neq q$, tal que $a_{sk} = a_{sq} = 1$, deve ser suprimida, uma vez que sendo $x_q = 1$, fazendo $x_k = 1$, teremos

$$\sum_{j \in N} a_{sj} x_j \geq 2, \qquad s \in M.$$

4. Recobrimento e Particionamento. Se $lin(s) \geq lin(p)$ (no sentido vetorial), para algum s, p, então lin(s) pode ser suprimida, uma vez que cada recobrimento da linha p, recobre também a linha s.

5. Particionamento. Para evitar um "super-recobrimento", junto com a supressão da linha s na redução anterior, cada coluna q, tal que $a_{sq} = 1$ e $a_{pq} = 0$, deve ser suprimida, uma vez que alguma coluna k com $a_{pk} = a_{sk} = 1$, deve ter $x_k = 1$, a fim de recobrir a linha p. Dessa forma, se $x_q = 1$, a linha s seria "super-recoberta".

6. Recobrimento e Particionamento. Se para algum conjunto Q de colunas e alguma coluna q, tivermos

$$\sum_{j \in Q} a_j = a_q \qquad \text{e} \qquad \sum_{j \in Q} c_j \leqslant c_q,$$

a coluna q pode ser suprimida. Na pior das hipóteses, as colunas de Q, desempenham as funções da coluna q, com a mesma eficiência. Em outras palavras, a coluna q e o conjunto de colunas Q, seriam redundantes.

7. Recobrimento. Se para algum conjunto Q de colunas e alguma coluna q, tivermos

$$\sum_{j \in Q} a_j \geqslant a_q \qquad \text{e} \qquad \sum_{j \in Q} c_j \leqslant c_q,$$

a coluna q pode ser suprimida, pelo mesmo motivo de (6).

3.6 - OS MODELOS DE RECOBRIMENTO E PARTICIONAMENTO E SUAS FORMAS CONTÍNUAS.

Algumas relações interessantes entre os modelos de recobrimento e particionamento e suas formas contínuas (modelos de programação linear), ajudam na construção e aperfeiçoamento de novos algoritmos para obtenção das soluções. Por exemplo, mudando as restrições $x_j \in \{0,1\}$, $\forall j \in N$, por $x_j \geqslant 0$, nos modelos dos (PCR) e (PCP), obtemos suas correspondentes formas contínuas (formas em programação linear) e vemos que o (PCR) tem uma solução binária viável se, e somente se, sua correspondente forma de PL, tem uma solução viável. Não há nada a ser provado no caso do (PCR) ter uma solução inteira e esta ser uma solução viável para seu correspondente PPL. A implicação contrária, fica clara a partir da seguinte construção:

Se $x' = (x_1, \ldots, x_n)$ é uma solução do PPL, obtemos uma solução

viável para o (PCR), fazendo

$$x_j = 0 \qquad \text{se} \qquad x'_j = 0 \quad \text{e}$$

$$x_j = 1 \qquad \text{se} \qquad x'_j > 0.$$

Agora, a existência de uma solução viável para o PPL de Particionamento, não implica necessariamente uma solução viável para o (PCP). Também, os PPL de recobrimento e particionamento correspondentes, onde ajuntamos ao conjunto de suas restrições, uma outra definida por $x_j \leqslant 1$, $\forall\, j \in N$, tem as mesmas soluções ótimas, quando estas existirem.

3.7 - ALGORITMO PARA A SOLUÇÃO DOS PROBLEMAS DE RECOBRIMENTO E PARTICIONAMENTO.

Para solução dos problemas de determinar os conjuntos recobrimento e particionamento mínimos, utilizaremos o algoritmo de enumeração implícita do CAPÍTULO II, acrescido de certas facilidades, como eliminação de certos testes, forma de armazenamento da matriz A e também as supressões de linhas e colunas da seção 3.5. Em problemas práticos, a matriz A é quase sempre esparsa (grande quantidade de entradas iguais a zero), o que sugere a utilização de algum processo de estocagem da matriz que mantenha só os elementos não nulos. Nestes modelos os elementos não nulos, são sempre iguais a 1 e os testes podem levar este fato em consideração. O algoritmo do CAPÍTULO II, junto com o processo que utilizamos para armazenar a matriz A, opera melhor em matrizes de baixa densidade (a densidade, sendo o número de 1's dividido pelo produto de m por n). Daremos maiores detalhes de seu funcionamento na descrição da construção do código, objeto central deste trabalho. A solução dos problemas de particionamento, é obtida pela transformação de cada restrição

$$\sum_{j \in N} a_{ij} x_j = 1,$$

em outras duas desigualdades

$$\sum_{j \in N} a_{ij} x_j \geqslant 1 \qquad \text{e} \qquad \sum_{j \in N} a_{ij} x_j \leqslant 1$$

Uma outra forma possível, é pela utilização da transformação proposta no teorema da seção 3.4.

CAPÍTULO IV

O EMPARELHAMENTO MÁXIMO E RECOBRIMENTO MÍNIMO EM GRAFOS NÃO ORIENTADOS.

4.1 - INTRODUÇÃO

Trataremos agora com algumas idéias dentro da teoria dos grafos, relacionadas com aquelas de recobrimento e particionamento em Programação Inteira. Na verdade, tratam-se de casos particulares dos modelos referidos acima. A matriz dos modelos de recobrimento e emparelhamento em um grafo não orientado, é a matriz de incidência desse grafo e portanto, tem apenas duas entradas não nulas em cada coluna.

4.2 - CONCEITOS BÁSICOS.

Seja

$$P = \{ i \mid i = 1, \ldots, k \}$$

um conjunto de pontos, vértices ou nós e seja

$$A = \{ (i,j) \mid i \in P,\ j \in P \}$$

um subconjunto de todos os pares não ordenados, formáveis com os elementos de P. Aos elementos de A, chamaremos arcos, eixos ou arestas.

Definição: Chamamos de "grafo não orientado" ao par $G = (P,A)$.

Dado um grafo $G = (P, A)$ não orientado, definimos o "grau" de $i \in P$ como

$gr_A(i)$ = número de arcos de A incidentes no vértice i.

Um "emparelhamento" de G é um subconjunto $E \subset A$, tal que $gr_E(i) \leqslant 1$, $\forall\, i \in P$.

Estamos definindo aqui o emparelhamento em um grafo não orientado, como um subconjunto dos arcos do grafo com a propriedade de que dois arcos quaisquer do subconjunto, não têm nenhum vértice em comum. Ou ainda com a propriedade de que cada vértice do grafo seja incidente no máximo a um único arco do subconjunto. Este é o emparelhamento dos nós por arcos. Dois a dois, os nós são ligados pelos arcos do subconjunto, sendo que cada par de nós é ligado por um único arco. Um conceito similar, é o de emparelhamento dos arcos pelos nós. Neste caso, temos um conjunto de nós, digo, um subconjunto, com a propriedade de que, dois a dois, os arcos do grafo são incidentes nos nós do subconjunto e cada nó é o ponto de conexão de somente dois arcos. Este conceito está ligado ao de "coeficiente de estabilidade interna", de um grafo. Nas nossas discussões, o termo emparelhamento se referirá ao dos nós pelos arcos. Todo grafo contém um emparelhamento, dado que

$$gr_{\phi}(i) = 0, \quad \forall\, i \in P.$$

Um caminho em um grafo G = (P,A), é uma sequência de arcos adjacentes (dois arcos são adjacentes quando têm um nó em comum) e um grafo é conexo, quando entre dois quaisquer de seus vértices, existe pelo menos um caminho. Sem restringir a generalidade, estamos supondo que o grafo G é conexo e se isto não acontece, os resultados abordados serão válidos para as suas componentes conexas (componentes conexas de um grafo são as classes definidas pela relação de equivalência: dois vértices i e j, são tais que ou i = j ou existe um caminho entre i e j).

E' é um "emparelhamento máximo" de G se

$$card(E') = \max \left\{ card(E) \mid E \text{ é emparelhamento de } G \right\}.$$

Se atribuirmos um peso, custo ou lucro c_j a um arco

$$a_j = (i,k) \in A,$$

podemos definir o peso, custo ou lucro de um emparelhamento E, como sendo

$$c(E) = \sum_{a_j \in E} c_j \qquad \text{(IV.1)}$$

Um emparelhamento E' é chamado de "emparelhamento máximo ponderado", se

$$c(E') \geqslant c(E), \quad \forall\, E \subset A. \qquad \text{(IV.2)}$$

Generalizando o que dissemos acima, podemos permitir que

$$gr_E(i) \leqslant b_i \qquad \text{(IV.3)}$$

onde b_i é um número inteiro positivo. Dado o vetor $b = (b_1, \ldots, b_m)$, chamamos E de um b-emparelhamento se

$$gr_E(i) \leqslant b_i, \quad \forall\, i \in E. \qquad \text{(IV.4)}$$

Se c e x são vetores de dimensão card(A), c_j é o peso, custo ou lucro associado ao arco $a_j = (i,k) \in A$ e $x_j = 1$ se o arco a_j está no b-emparelhamento E e $x_j = 0$, caso contrário, o problema de encontrar o b-emparelhamento máximo ponderado, pode ser expresso da seguinte forma:

$$\text{maximizar } z = \sum_{a_j \in A} c_j x_j, \qquad \text{(IV.5)}$$

$$\text{sujeito a} \sum_{a_j \in A} q_{ij} x_j \leqslant b_i, \; i \in P \qquad \text{(IV.6)}$$

$$x_j \in \{0,1\}, \qquad \text{(IV.7)}$$

onde $Q = (q_{ij})$ é a matriz de incidência do grafo $G = (P,A)$ e $q_{ij} = 1$ se o ar

co a_j é incidente no nó $i \in P$ e $q_{ij} = 0$, caso contrário. Se $b = (1,\ldots,1)$, temos o caso do emparelhamento máximo ponderado e se $c = (1,\ldots,1)$, simplesmente o problema do emparelhamento máximo.

Muitos problemas contínuos exibem soluções inteiras (por exemplo, o problema do transporte, da atribuição). A condição suficiente para isso é que a matriz do modelo seja totalmente unimodular (Uma matriz A é totalmente unimodular se, e somente se, toda submatriz quadrada de A, tem determinante igual a -1, 0 ou 1. Se A é a matriz de um problema de programação linear, toda base B, já que B^{-1} existe e A é totalmente unimodular, tem $\det(B) \in \{-1, 1\}$). No caso do problema de emparelhamento máximo, poderia acontecer o mesmo, o que dispensaria a restrição de que as variáveis x_j fossem inteiras e a solução do problema poderia ser encontrada utilizando o método simplex (o problema seria resolvido como um PPL). Entretanto isso não acontece. Vejamos um exemplo que mostrará a necessidade de (IV.7) ou a de que os x_j's sejam inteiros ou ainda a necessidade de que o problema seja resolvido por um processo que produza soluções desse feitio.

Exemplo:

Consideremos o grafo da figura IV.1

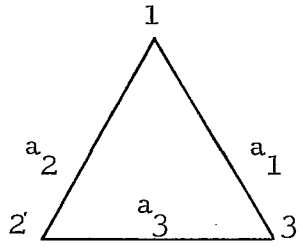


Figura IV.1

Cada um dos emparelhamentos $\{a_1\}$, $\{a_2\}$, e $\{a_3\}$ é um emparelhamento máximo (ótimo). O problema de Programação Inteira, que produzirá como solução estes emparelhamentos, é

$$\text{maximizar } z = x_1 + x_2 + x_3$$

$$\text{sujeito a} \quad \begin{aligned} x_1 + x_2 &\leqslant 1 \\ x_1 + x_3 &\leqslant 1 \\ x_2 + x_3 &\leqslant 1 \end{aligned}$$

$$x_j \in \{0,1\}, \quad j = 1, 2, 3.$$

Se trocarmos as restrições $x_j \in \{0,1\}$, $j = 1, 2, 3$, por $x_j \geqslant 0$, e submetermos o problema ao método simplex, a única solução ótima é dada por

$$x_1 = x_2 = x_3 = \frac{1}{2}$$

e a matriz $Q = (q_{ij})$ do modelo, dada por

$$Q = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

é tal que $\det(Q) = -2$. Por este exemplo vemos que a matriz do modelo de emparelhamento, não é necessariamente unimodular, digo, totalmente unimodular.

Agora vejamos uma situação em que podemos garantir a unimodularidade total da matriz do modelo.

4.3 - EMPARELHAMENTO EM GRAFOS BIPARTIDOS.

Um grafo $G = (P,A)$ é bipartido se existem conjuntos P_1 e P_2, tais que

$$P_1 \cup P_2 = P \quad \text{e} \quad P_1 \cap P_2 = \phi$$

e cada arco de A é incidente em um nó de P_1 e em um nó de P_2.

A matriz $Q = (q_{ij})$ de incidência de um grafo bipartido, é totalmente unimodular, posto que a partição da definição de grafo bipartido se

enquadra nas hipóteses do

Teorema: Uma matriz inteira $Q = (q_{ij})$, com $q_{ij} \in \{0, 1, -1\}$, para todo i e todo j, é totalmente unimodular, se

a. Não mais do que dois elementos distintos de zero aparecem em cada coluna de Q.

b. As linhas de Q podem ser particionadas em dois conjuntos K_1 e K_2, tais que

i) Se uma coluna contém dois elementos distintos de zero com o mesmo sinal, um elemento está em K_1 e o outro em K_2.

ii) Se uma coluna contém dois elementos distintos de zero com sinais opostos, ambos estão no mesmo conjunto.

Demonstração: (por indução)

Tomando as submatrizes de Q constituidas de um único elemento, temos que os possíveis valores dos determinantes destas submatrizes, são -1, 0, 1. Supondo que os valores dos determinantes das submatrizes de Q de ordem p - 1, são também iguais a -1, 0, 1 (hipótese de indução) e que Q' é uma submatriz de ordem p, temos:

1. Se Q' tem uma coluna nula, det(Q') = 0.

2. Se existe uma coluna com um único elemento distinto de zero, expandimos o det(Q') por esta coluna (teorema de Laplace) e pela hipótese de indução, $\det(Q') \in \{-1, 0, 1\}$.

3. Se cada coluna de Q' tem dois elementos não nulos, temos por (i) e (ii) que

$$\sum_{i \in K_1} q_{ij} = \sum_{i \in K_2} q_{ij}. \qquad \text{(IV.8)}$$

Agora, representando por q_i a i-ésima linha de Q', temos por (IV.8), que

$$\sum_{i \in K_1} q_i = \sum_{i \in K_2} q_i \qquad \text{(IV.9)}$$

ou que

$$\sum_{i \in K_1} q_i - \sum_{i \in K_2} q_i = 0, \qquad \text{(IV.10)}$$

cujo significado é o de que as linhas de Q' são linearmente dependentes e portanto, det(Q') = 0.

O teorema continua valendo, ainda que um dos cunjuntos, K_1 ou K_2 seja vazio.

O clássico problema da atribuição pode ser interpretado como um problema de emparelhamento ponderado num grafo bipartido.

4.4 - O PROBLEMA DO RECOBRIMENTO EM UM GRAFO

Dado um grafo G = (P,A), não orientado, um subconjunto $R \subset A$ é denominado um "recobrimento" de G se $gr_R(i) \geqslant 1$, $\forall i \in P$.

Um recobrimento R' é chamado de recobrimento mínimo se

$$\text{card}(R') = \min \left\{ \text{card}(R) \mid R \text{ é recobrimento de } G \right\}.$$

Aqui estamos tratando do recobrimento dos nós pelos arcos de um grafo e como no caso do emparelhamento, existe um conceito similar de recobrimento dos arcos pelos nós. Como antes, lembramos que o termo recobrimento, se refere ao dos nós pelos arcos. Também recobrimentos ponderados e b-recobrimentos, são definidos como no caso do emparelhamento.

O problema de encontar o recobrimento mínimo ponderado de um grafo G, é descrito por

$$\text{minimizar} \quad z = \sum_{a_j \in A} c_j x_j \tag{IV.11}$$

$$\text{sujeito a} \quad \sum_{a_j \in A} q_{ij} x_j \quad b_i, \quad i \in P \tag{IV.12}$$

$$x_j \in \{0,1\}, \qquad a_j \in A \tag{IV.13}$$

$$q_{ij} \in \{0,1\}, \quad i \in P, \; a_j \in A \tag{IV.14}$$

Existem relações interessantes entre o emparelhamento máximo e o recobrimento mínimo em um grafo não orientado. Vejamos algumas.

4.5 - RELAÇÕES ENTRE EMPARELHAMENTOS E RECOBRIMENTOS EM UM GRAFO NÃO ORIENTADO.

Antes de introduzir os resultados que traduzem algumas relações entre emparelhamentos máximos e recobrimentos mínimos em grafos não orientados, lembramos que estamos tratando com grafos sem laços e tais que entre dois vértices quaisquer existe no máximo um arco (a um grafo deste tipo, chamamos de "grafo simples"). Também, os conceitos de "nó saturado por um emparelhamento" e "emparelhamento perfeito", serão dados agora.

Um nó $i \in P$ é dito "saturado" pelo emparelhamento E de um grafo $G = (P,A)$, se um arco de E é incidente no nó i.

Um emparelhamento que satura todos os nós de G, é chamado de "emparelhamento perfeito". Naturalmente, um emparelhamento perfeito, é um em

parelhamento máximo.

Dado um grafo G = (P,A), se E é um emparelhamento de G, representamos por Sat(E) o conjunto dos vértices de P saturados por E e por $C_{Sat(E)}$ o conjunto dos vértices de P não saturados por E. (Relativamente a um emparelhamento E, os conjuntos Sat(E) e $C_{Sat(E)}$, são complementares).

Nos modelos definidos por (IV.5), (IV,6), (IV.7), (IV.11), (IV.12), (IV.13) e (IV.14), tomemos os vetores b e c, como b = (1,...,1) e c = (1,...,1). Em casos assim, dado um grafo G = (P,A) que tem E como emparelhamento máximo e R como um recobrimento mínimo, temos

$$card(E) \leqslant card(R) \qquad (IV.15)$$

A relação (IV.15), permite o seguinte:

Pelo acréscimo de arcos a um subconjunto que seja um emparelhamento máximo, obtemos um recobrimento mínimo e pela supressão de arcos de um subconjunto que seja um recobrimento mínimo, obtemos um emparelhamento máximo. Formalizando melhor, temos

i. Seja R um recobrimento mínimo de um grafo não orientado G = (P,A). Para cada nó $i \in P$, tal que $gr_R(i) > 1$, remova arcos incidentes em i, em número igual a $gr_R(i) - 1$. O conjunto resultante de arcos, é um emparelhamento máximo de G.

ii. Seja E um emparelhamento máximo de um grafo não orientado G = (P,A). Se adicionarmos um arco a cada nó i, tal que $gr_E(i) = 0$, o conjunto de arcos resultante, é um recobrimento mínimo de G.

Agora, dado um grafo G = (P,A) que tem E como um emparelhamento máximo e R como um recobrimento mínimo e observando que

$$card(Sat(E)) = 2card(E),$$

temos

$$card(E) + card(R) = card(P). \qquad (IV.16)$$

De fato,

$$R = E \bigcup \left\{ a_j = (i,k) \in A \mid k \in C_{Sat(E)} \right\} \quad e$$

$$card(R) = card(E) + card\left(\left\{ a_j = (i,k) \in A \mid k \in C_{Sat(E)} \right\} \right) -$$

$$- card\left(E \bigcap \left\{ a_j = (i,k) \in A \mid k \in C_{Sat(E)} \right\}\right).$$

$$card(R) = card(E) + card(P) - 2card(E)$$

$$card(R) = card(P) - card(E),$$

de onde tiramos a relação (IV.16).

4.6 - OBSERVAÇÕES ADICIONAIS.

Naturalmente, estamos interessados na máxima aplicabilidade do código que desenvolvemos para solução de vários problemas de natureza combinatória. Com isso em mente, abordamos também os casos do emparelhamento máximo e do recobrimento mínimo em grafos não orientados, suas relações (algumas e talvez as mais simples) e o fizemos de forma superficial, mas o suficiente para identificarmos a conveniência de se aplicar o algoritmo desenvolvido no CAPÍTULO II, na solução dos problemas modeláveis com estes recursos. Como vimos, nem sempre a matriz destes modelos é totalmente unimodular e portanto não temos como aplicar o método simplex para a obtenção das soluções discretas destes problemas. Este enfoque, aparece como um tratamento alternativo, inclusive para problemas como o da atribuição e do transporte, tratados freqüentemente dentro de um outro contexto. Tratados assim, com recursos da teoria dos grafos, certamente cooperam para que esta teoria se torne progressivamente mais atraente.

CAPÍTULO V

CÓDIGO PARA A SOLUÇÃO DOS PROBLEMAS DE PROGRAMAÇÃO INTEIRA BIVALENTE, DE DETERMINAÇÃO DOS CONJUNTOS RECOBRIMENTO E PARTICIONAMENTO MÍNIMOS, EMPARELHAMENTO MÁXIMO E RECOBRIMENTO MÍNIMO EM UM GRAFO NÃO ORIENTADO.

5.1 - INTRODUÇÃO.

Aqui, procuraremos mostrar como desenvolvemos o código para a solução dos problemas referidos no título. Tentaremos orientar um possível usuário na utilização do programa para a solução de um problema cujo modelo seja um dos mencionados acima.

5.2 - A LINGUAGEM UTILIZADA.

O programa foi desenvolvido em ALGOL/B6700. Algol é uma linguagem de programação, estruturada em bloco, ORGANICK[45], própria para representação de algoritmos complexos, dada sua eficiência em otimizá-los e processá-los com recurso de alocação dinâmica de memória, KNUTH[35], i.e., com recursos de algoritmos que reservam e liberam dinamicamente, blocos de memória de tamanho variável. Estas propriedades algoritmicas da linguagem, atendem melhor nosso trabalho e seu processamento é o mais adequado no computador que utilizamos.

5.3 - ASPECTOS BÁSICOS DO PROGRAMA.

Uma rotina de entrada e saída faz todo o gerenciamento da leitura dos dados, iniciação das variáveis, chamada das rotinas que fazem a busca da solução do problema em questão e emite os resultados e/ou outras informações referentes ao processamento. Existem duas procedures para a enumeração implícita das pseudo-soluções parciais: uma para os problemas de programação inteira bivalente em que a matriz do modelo não tem forma ou estrutura especial (por exemplo, não é esparsa, os elementos são inteiros quaisquer, iguais a zero, positivos ou negativos) e o vetor b também é constituído de inteiros quaisquer; a outra já leva em conta a estrutura especial da matriz A, onde os elementos são todos iguais a zero ou um, é esparsa e o vetor b tem todos os elementos iguais a 1. A forma dos testes e todo o tratamento dos dados têm características bem distintas daquelas do primeiro caso. A procedure ENUXIMPXRECXPAR, faz a busca das soluções dos problemas de recobrimento e particionamento mínimos, emparelhamento máximo e recobrimento mínimo em um grafo não orientado. A rotina ENUMXIMPL, se encarrega da enumeração implícita das pseudo-soluções de um problema inteiro bivalente. Com o que dissemos no início do CAPÍTULO II, poderíamos resolver qualquer problema de programação inteira, mas trataríamos desnecessariamente com problemas que dariam origem a um grande número de variáveis e estes são solúveis a custo de menor esforço, utilizando outras técnicas mais adequadas.

5.4 - DESCRIÇÃO SUCINTA DAS PRINCIPAIS ROTINAS DO PROGRAMA E SUAS PRINCIPAIS VARIÁVEIS LOCAIS OU PASSADAS COMO PARÂMETROS.

Faremos uma descrição na ordem em que o programa é impresso e pela ordem em que aparecem no programa, as procedures fundamentais, são:

DETXMIN: na solução de um problema de programação linear de minimização, o

algoritmo simplex faz a determinação da coluna pivô (ou da variável que deve entrar na base), através desta procedure.

PROXLIN: rotina para solução de um problema de programação linear pelo método simplex revisado. Esta procedure é utilizada para geração de restrições substitutas, calculando as soluções, ou melhor os valores das variáveis duais relativas às restrições de subproblemas do problema de programação inteira em questão, como explicado no CAPÍTULO II. Construida com esse objetivo, a rotina não está implementada para a solução de um problema em que a origem não seja uma solução viável. A forma dos problemas submetidos a essa rotina, dispensa os recursos para criação de uma solução inicial (por exemplo, método das duas fases). O tableau simplex APL, passado como parâmetro, é armazenado como um arranjo de uma única linha ou coluna e a referência a um elemento de coordenadas I e J (linha I, coluna J), é feita através de

$$APL(I \times DECOL + J \times DELIN),$$

onde DECOL é a distância na memória entre dois elementos consecutivos de uma coluna (é um passo de linha) e DELIN, a distância entre dois elementos consecutivos de uma linha (é um passo de coluna). Neste caso, os índices I e J funcionam como apontadores. O tableau simplex tem m + 1 linhas e n + 1 colunas, incluindo os custos e o vetor constante e se ele é armazenado na forma de uma única coluna, temos:

$$DECOL = 1 \qquad e \qquad DELIN = m + 1.$$

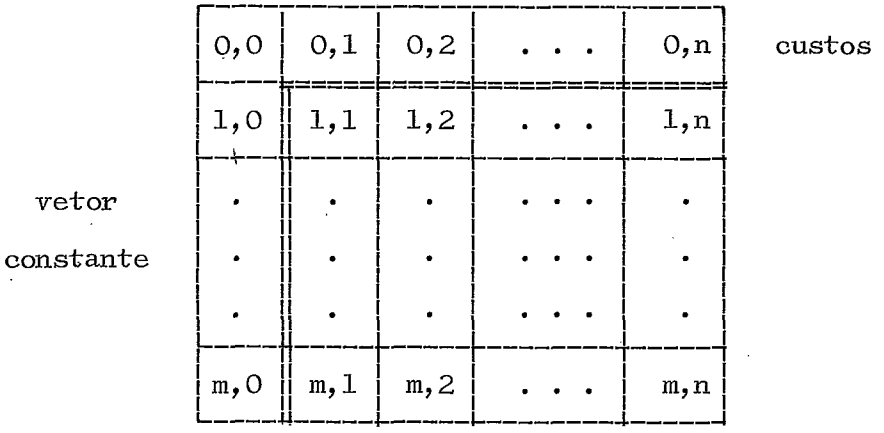
E se é armazenado na forma de uma única linha, temos:

$$DECOL = n + 1 \qquad e \qquad DELIN = 1.$$

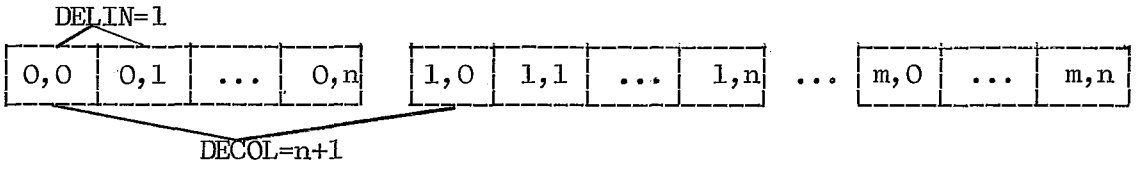
A figura V.1, é uma ilustração que explica melhor a função destas variáveis. As outras variáveis passadas como parâmetros, são:

INDVB: vetor que dá os índices das variáveis básicas.

INDVN: o mesmo para as variáveis não básicas.



arranjo em forma de linha:



arranjo em forma de coluna:

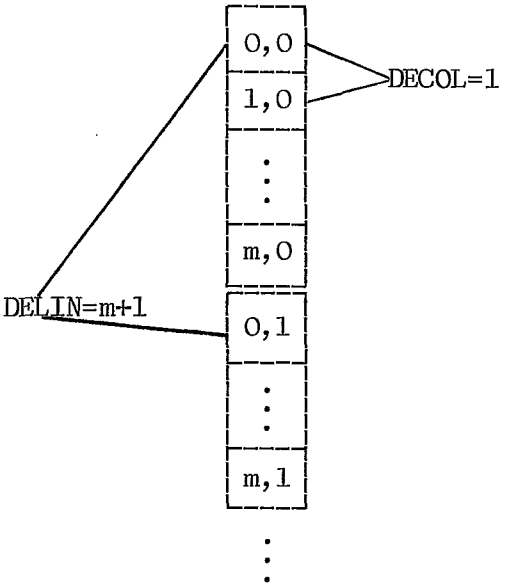


Figura V.1

Esquema de armazenamento do tableau APL.

ZA: é o valor da função objetivo do problema inteiro, associado à atual pseudo-solução parcial.

ZXMENOR: é o melhor valor da função objetivo do problema inteiro, conhecido até aqui.

A rotina INVERTE, local à procedure PROXLIN, cuida da inversão da matriz da base a cada passo. As outras "principais" variáveis, são:

COMPLETOU: é uma variável lógica (booleana), que mantém o processamento no laço principal da rotina enquanto seu estado é "FALSE".

SALTA: é também uma variável lógica para controle dentro do programa.

TP: indica se o problema não tem solução viável, se o problema tem ou não solução finita.

IX, IY: vetores com os índices, respectivamente, das colunas e linhas (ou índices das variáveis não básicas e básicas) admissíveis para uma troca.

IZ: vetor auxiliar de IX e IY.

A presença do processamento no laço principal de PROXLIN, é mantida enquanto a variável lógica COMPLETOU está no estado FALSE e a expressão

$$APL(0) + ZXMENOR - 1 - ZA \qquad (V.1)$$

for não negativa. Nela, APL(0) é o valor da função objetivo do problema de programação linear em questão. Quando esta expressão fica negativa, como mostrado no CAPÍTULO II, a presente pseudo-solução parcial não tem complemento viável que produza uma cota superior melhor do que a atual ZXMENOR. Portanto, o algoritmo poda a presente pseudo-solução parcial, i. e., enumera a atual pseudo-solução parcial e enumera implicitamente todos os seus complementos. Naturalmente, quando a saída da procedure PROXLIN é feita pela negatividade de (V.1), não ocorre geração de restrições substitutas, dado que estas

são, nesse caso, binário inviáveis.

A rotina SOLXDUAL, completa a preparação do problema de programação linear que fornecerá os valores das variáveis duais, dimensionando convenientemente os arranjos, atualizando (reiniciando) algumas variáveis e chamando a procedure PROXLIN. De acordo com os resultados dados por PROXLIN, ela cria os coeficientes para posterior geração das restrições substitutas por uma outra rotina.

A procedure RESXSUB, inicia a preparação do problema de programação linear que fornecerá os valores das variáveis duais, no caso dos modelos de recobrimento e particionamento mínimos, emparelhamento máximo e recobrimento mínimo em um grafo não orientado. No caso dos problemas de programação discreta bivalente, o mesmo papel é desempenhado por RESTRICXS. A necessidade destas duas rotinas está principalmente no fato das matrizes dos modelos serem armazenadas de forma distinta, como veremos. RESXSUB e RESTRICXS têm em comum (chamam) a procedure SOLXDUAL. Suas principais variáveis são:

U: vetor para guardar os valores das variáveis duais, soluções do problema de programação linear, usadas na geração das restrições substitutas

$$u(b + Az) + (z_{min} - cx) \geqslant 0.$$

AVANÇA: variável inteira que recebe e transmite para futuras decisões, os valores da expressão

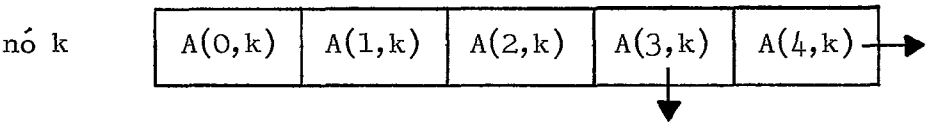
$$APL(0) + ZXMENOR - 1 - ZA.$$

Como dito anteriormente, se AVANÇA é não negativa, é gerada a restrição substituta e com base nos testes de viabilidade binária do CAPÍTULO II, novos avanços na árvore das pseudo-soluções parciais, são produzidos pela fixação de variáveis livres nos níveis zero ou um.

GERA: variável lógica. Se GERA é FALSE, a restrição substituta não é gerada, o algoritmo poda a pseudo-solução parcial.

AS, BS: arranjos que guardam os coeficientes das restrições substitutas mantidas para serem submetidas ao teste de viabilidade binária. Podem ser mantidas tantas restrições substitutas quantas quisermos e a cada passo, nova restrição é adicionada ao conjunto, enquanto outras são abandonadas. Em RESTRICXS, são mantidas sempre duas restrições substitutas e a cada passo a mais antiga é abandonada no momento em que se adiciona a última restrição gerada. A título de ilustração, mantivemos em RESXSUB, quatro restrições substitutas embora nossa crença seja de que devam ser mantidas apenas duas.

ENUXIMPXRECXPAR, como já mencionamos, é a parte do código que resolve o problema no caso da matriz $A = (a_{ij})$ ser tal que $a_{ij} \in \{ 0,1 \}$, para todo i e todo j. Esta matriz é esparsa em problemas práticos e aproveitamos esse fato para armazenar somente os elementos não nulos e fizemos isto utilizando uma estrutura em lista, KNUTH[35], i.e., um esquema de armazenamento em que a ordem lógica dos dados é determinada por apontadores (ponteiros). Um elemento k da lista (um só) tem o seguinte aspecto:



A(0,k) contém o valor do elemento e é dispensável nos casos dos modelos em que esse valor é sempre um. Mantivemos no programa o elemento armazenado, dado ao modo particular, como mostraremos mais tarde, de resolver os problemas do conjunto particionamento. Nesses problemas alguns elementos são -1. A(1,k) dá a linha do elemento, A(2,k) a coluna, A(3,k) aponta para o próximo elemento da coluna e A(4,k) para o próximo da linha. De resto, o algoritmo de enumeração implícita é o do CAPÍTULO II, cujo fluxograma também está representado lá. A rotina ENUMXIMPL, para solução de qualquer problema de programação discreta 0-1, usa o mesmo algoritmo e a matriz do medelo é armazenada naturalmente como um arranjo qualquer de m linhas e n colunas. Mantivemos para as duas rotinas os mesmos nomes de suas principais variáveis, como

PERMXLAÇO: é a variável lógica que mantém o processamento no laço principal da procedure enquanto seu estado for TRUE.

V: conjunto dos índices de variáveis livres candidatos a participarem da próxima pseudo-solução parcial.

W: contém a cada passo a pseudo-solução parcial e foi implementado como uma pilha onde estão os nós da árvore das pseudo-soluções parciais. Seu tamanho (quantidade de nós) varia naturalmente com o processo de enumeração: acréscimo ou decréscimo dos índices fixados, poda das pseudo-soluções parciais.

XSOLXATUAL: guarda a melhor solução viável X.

B: vetor constante.

C: vetor dos custos.

ZERO: vetor que indica quais variáveis foram fixadas no nível zero, com base no teste de viabilidade binária.

Cada uma das duas procedures, ENUXIMPXRECXPAR e ENUMXIMPL, tem uma rotina, SUBLIXCOMPL, local, para o processo de poda das pseudo-soluções parciais. Ela passa ao complemento lógico do último elemento ainda não complementado, digamos, mais acima na pilha W e abandona todos os elementos que já passaram ao complemento lógico, acima deste último complementado, conforme discutido no CAPÍTULO II. Além disso, ENUXIMPXRECXPAR tem as seguintes variáveis

PRIXLIN: cabeça de lista. Aponta para o primeiro elemento de cada linha.

PRIXCOL: o mesmo para cada coluna.

LINXSEG: apontador para a linha seguinte a uma dada linha. Sua utilidade aparece após as reduções que suprimem linhas e colunas da matriz ,

conforme CAPÍTULO II.

As procedures de enumeração implícita, foram implementadas utilizando uma pilha de nós disponíveis, para facilitar a programação e visualização daquilo que o algoritmo faz. ENUMXIMPL, tem um recurso opcional para a criação de uma solução inicial, conforme mencionamos no CAPÍTULO II. As experiências que desenvolvemos mostraram que em alguns casos ela acelera o processo de enumeração, em outros, é indiferente e nos casos dos modelos de recobrimento e particionamento, desaconselhamos seu uso. Na ENUXIMPXRECXPAR, através do teste de $BALAS^2$, o algoritmo acrescenta um único índice em cada avanço no processo de enumeração (criação de descendentes) e na ENUMXIMPL, acrescenta vários, quando isso é possível. Sempre que uma solução for viável para as restrições substitutas, testamos sua viabilidade para as restrições do problema e no caso de uma solução inviável, tentamos vencer a inviabilidade através dos testes do algoritmo de enumeração implícita comum (sem restrições substitutas), utilizando as restrições originais do problema. Pois sabemos que se x^o é uma solução viável para $Ax \geqslant b$, ela também é viável para $uAx \geqslant ub$, mas o caso contrário, não é necessariamente verdadeiro (veja exemplo do CAPÍTULO II).

A procedure REDUÇÕES, executa as supressões de linhas e colunas, de acordo com os critérios mencionados no CAPÍTULO III. Suas rotinas locais são DESATIVAXLINHA e DESATIVAXCOLUNA, cujas funções, os próprios nomes indicam. De suas variáveis, destacamos

RECOBRE: indica se o problema para o qual estão sendo feitas as reduções, é de recobrimento.

PARTICI: a mesma coisa para um problema de particionamento.

A procedure AJUSXPONT, cuida da iniciação das variáveis no caso dos problemas em que a matriz é 0-1. ATIVXPONT, ativa (liga) os ponteiros aos dados à medida que estes são lidos pela procedure ENTRADAXSAIDA. As rotinas LIGAXPONTXLINHA e LIGAXPONTXCOLUNA, são locais à ATIVXPONT e a função de

las está indicada no nome. REAJXPONT atualiza o apontador LINXSEG após a supressão de linhas e colunas da matriz do modelo. A rotina ESCREXDADOS, escreve a matriz dos modelos 0-1, na sua forma normal. ESCREXRESUL, dá os resultados de todos os problemas 0-1. Na procedure ENTRADAXSAIDA, são tomadas as decisões sobre os caminhos a serem seguidos para a solução do problema, através das variáveis

ENUMERA: se for um problema de programação discreta 0-1, qualquer.

RECOBRE: se for de recobrimento.

PARTICI: se for de particionamento.

EMPAREL: de emparelhamento.

RECXGRA: de recobrimento em grafos.

As reduções são opcionais, mas caso não seja indicado o contrário, o programa sempre as executa regularmente quando possível. Em problemas de pequeno porte, as experiências nos indicaram que são dispensáveis, dado que o tempo despendido com elas é muitas vezes suficiente para a solução do problema. O programa foi exaustivamente testado com problemas de dificuldade reconhecida e apresentou excelentes resultados quando as restrições substitutas foram introduzidas. Um dos testes feitos, mostrou que com um tempo de processamento de cinco horas, o algoritmo sem a implementação de restrições substitutas, não conseguiu obter a solução ótima de um problema de 50 variáveis e 5 restrições. Já implementado com as restrições substitutas, a solução foi obtida com 19 minutos de processamento. Uma outra tentativa foi a geração de uma origem, obtida a partir da solução contínua (por programação linear). Esta se revelou uma heurística pouco eficiente nos casos em que a solução contínua está perto da solução inteira ótima. A versão definitiva do programa não está implementada com esse recurso.

5.5 - CONCLUSÕES.

As experiências adquiridas com testes exaustivos do código construído, nos autorizam dizer que para problemas a partir de um certo tamanho, o processo de enumeração implícita de pseudo-soluções parciais em Programação Inteira bivalente, sem outros recursos adicionais, é um recurso pouco poderoso. Entretanto, com a implementação de restrições substitutas, o processo torna-se bastante eficiente, principalmente nos casos em que a matriz do modelo é do tipo 0-1 e é armazenada como mostramos, utilizando uma estrutura adequada. Relativamente às reduções, desaconselhamos seu uso para problemas pequenos. A geração de uma solução inicial como a que foi implementada na rotina ENUMXIMPL, em alguns casos acelera o processo de enumeração , mas desaconselhamos seu uso para a maioria dos problemas de recobrimento.

5.6 - COMO UTILIZAR O PROGRAMA.

Daremos agora as orientações para utilização do programa.

O primeiro cartão de dados é o da dimensão do problema. Os valores M e N, nesta ordem, devem ser perfurados no formato 2I3, o primeiro campo, começando na coluna 1.

O segundo cartão, informa o tipo do problema:

Enumeração Implícita:	perfurar	1	na	coluna	1
Recobrimento:	"	2	"	"	2
Particionamento:	"	3	"	"	3
Recobrimento em grafos:	"	4	"	"	4
Emparelhamento em grafos:	"	5	"	"	5.

Nos problemas de Recobrimento, Particionamento e Recobrimento

em grafos, as reduções (eliminação de linhas e colunas), são sempre executadas, entretanto, se isto não for desejado, no mesmo cartão que indica o tipo do problema, deve ser perfurado na coluna 15, um número maior que zero e menor que dez.

No caso de um problema de enumeração implícita, o próximo cartão será o que indica:

Tipo de restrição:

Perfurar 1 na coluna 1 se for do tipo $\leqslant$. Obs.: o programa não está preparado para resolver problemas com restrições de igualdade estrita (=) somente $\geqslant$ ou $\leqslant$.

Se vai gerar uma solução inicial:

Perfurar 1 na coluna 2.

Se o tipo do problema for de maximização:

Perfurar 1 na coluna 3.

Para qualquer dos cinco tipos de problemas, os próximos dados serão: CUSTOS, MATRIZ DO MODELO e VETOR CONSTANTE, nesta ordem.

No caso de um problema de enumeração implícita, os CUSTOS, MATRIZ DO MODELO e o VETOR CONSTANTE, são perfurados no formato 16I5, o primeiro campo começa na coluna 1.

No caso dos outros problemas, temos o seguinte:

Os CUSTOS: O formato é 16I5, o primeiro campo começando na coluna 1.

A MATRIZ DO MODELO: Cada cartão informa a linha e a coluna de um único elemento. O formato é 2I4. O último cartão para leitura da matriz do modelo, é um CARTÃO BRANCO. O primeiro campo do formato começa na coluna

1.

O VETOR CONSTANTE: O formato é 16I5, o primeiro campo começando na coluna 1.

O último cartão de dados será um CARTÃO BRANCO para informar que não será lido nenhum outro problema.

Lembramos que a forma das restrições de um problema para ser resolvido pelo método de enumeração implícita é

$$b + Ax \ (\geqslant \text{ ou } \leqslant) \ 0,$$

i.e., com o vetor constante, no lado esquerdo da desigualdade.

Se o problema é de um dos outros tipos, as restrições são

$$Ax \ (\geqslant \text{ ou } \leqslant) \ b,$$

ou seja, com o vetor constante, no lado direito da desigualdade.

CAPÍTULO VI

APLICAÇÃO: CATALOGAÇÃO E ALOCAÇÃO DE TRIPULAÇÕES EM LINHAS (ROTAS) AÉREAS.

6.1 - INTRODUÇÃO.

Para ilustrar e mostrar a aplicabilidade do código que desenvolvemos, vamos propor um problema de alocação de tripulações em rotas aéreas e dar as linhas básicas da solução. Não será um problema que envolverá a complexidade natural das operações dessa natureza, mas será suficiente para recomendar um encaminhamento ou uma metodologia possível para tratar situações como a que apresentaremos.

6.2 - ASPECTOS GERAIS E FORMULAÇÃO DO PROBLEMA.

Tendo uma companhia aérea estabelecido uma tabela de horários para seus vôos (as tarefas), surge o problema de como preencher os horários ou os segmentos de vôos (um vôo ou um conjunto de vôos consecutivos) com as tripulações disponíveis (recursos), satisfazendo os regulamentos da empresa, do local onde ela opera, as condições contratuais dos membro das tripulações, etc., de tal forma ainda que o custo total de operação seja mínimo. Com isso temos um problema cujo modelo genérico tem a forma daqueles de Programação Matemática, i.e., otimizar (minimizar) uma função de custo, sujeito a uma ou mais restrições. Particularmente, os modelos mais frequentes, são de programação discreta bivalente, pois são de natureza essencialmente combinatória, queremos recobrir os horários (tarefas) com as tripulações (recursos disponíveis), como no CAPÍTULO III.

O problema de otimizar a alocação de tripulações a vôos de uma companhia aérea, tem sido objeto de vários estudos e sua solução, às vezes parcialmente satisfatória, tem sido conseguida por diferentes enfoques (exatos, heurísticos), pelas diferentes companhias. ARABEYRE[1], mostra os diferentes tratamentos dados ao problema pelas diferentes empresas e agrupa em alguns itens gerais as principais fases de sua solução.

No tratamento do problema, uma das questões iniciais, é a determinação do período para o qual o planejamento é válido. Chamando-o de "período de planejamento", ele pode ser de um dia, uma semana ou mesmo de um mes, o que influi naturalmente no tamanho do problema (número de variáveis e/ou restrições). Uma outra situação que o otimizador enfrenta no início, é a da determinação da "unidade de planejamento". Quase sempre, esta unidade é uma "rotação" e é definida como uma viagem de tamanho fixo (tempo ou número de segmentos de vôos), feita por uma tripulação, partindo de sua base (local de residência ou formação da tripulação) e incluindo o retorno ao ponto de partida. Também pode ser considerada como unidade de planejamento, o menor segmento de vôo, ligando duas cidades. Assim, o problema é o da seleção de um conjunto de rotações com tripulações associadas a elas, de modo a cobrir cada segmento de vôo pelo menos uma vez, a custo mínimo.

Imaginando um conjunto de n caminhos ou n rotações (cada caminho possivelmente constituido de um ou vários segmentos de vôos) e um conjunto de m vôos possíveis, como escolher dentre os caminhos, alguns, de tal forma que todos os vôos sejam aproveitados e o custo de operação minimizado? A resposta pode ser dada pela solução do problema clássico de determinar o conjunto recobrimento (ou particionamento) de custo mínimo. A matriz $A = (a_{ij})$ do modelo é gerada (falaremos mais tarde como), de tal forma que cada uma de suas colunas seja uma rotação possível e $a_{ij} = 1$ se o vôo i fizer parte da rotação (caminho de ida e volta) j e $a_{ij} = 0$, caso contrário. Se $c = (c_j)$ é o vetor dos custos (c_j é o custo do caminho/rotação j), podemos buscar como no CAPÍTULO III, a solução, abstraindo de outras restrições possivelmente existentes, através do modelo

$$\text{minimizar } z = \sum_{j \in N} c_j x_j, \qquad \text{(VI.1)}$$

$$\text{sujeito a} \sum_{j \in N} a_{ij} x_j \geqslant 1, \quad i \in M \qquad \text{(VI.2)}$$

$$x_j \in \{0,1\}, \qquad j \in N \qquad \text{(VI.3)}$$

$$a_{ij} \in \{0,1\}, \qquad i \in M, \quad j \in N. \qquad \text{(VI.4)}$$

Na solução, $x_j = 1$ se o caminho/rotação j é escolhido e $x_j = 0$ caso contrário. Quando as rotações são escolhidas de tal forma que um segmento de vôo é coberto mais de uma vez, temos uma situação operacional freqüente em que uma tripulação viaja como passageira para retornar à base ou assumir algum outro segmento de vôo.

Como já dissemos, uma rotação possível será representada por uma coluna da matriz $A = (a_{ij})$. Os vôos (segmentos de vôos), serão as linhas. Assim, a matriz A pode ser considerada como um "catálogo" de todas as rotações possíveis e consideradas. Após a geração da matriz, procedemos à sua redução à correspondente matriz de tamanho, digamos, manuseável como também à eliminação de linhas e colunas, conforme mostramos no CAPÍTULO III, já na fase de otimização.

Algumas companhias juntam ao conjunto das restrições naturais, uma série de outras restrições que limitam a quantidade de trabalho de uma base de tripulações. Estas são conhecidas como "restrições de base".

6.3 - O GERADOR DA MATRIZ DAS RESTRIÇÕES (MATRIZ CATÁLOGO).

O problema da catalogação e alocação das tripulações tem pos-

sivelmente como sua principal fase, a questão da geração da matriz das restrições e naturalmente cada companhia, levando em conta as suas características e as do lugar onde ela opera, tem o seu gerador de matriz. Este gerador é um programa implementado com características básicas da empresa, como quantidade de aeronaves (consideradas como unidade de produção), mão de obra disponível, demanda e com características do local de operação ou então esses elementos são fornecidos como dados ao gerador. A geração da matriz é feita, levando em conta também o período e a unidade de planejamento, as condições contratuais dos empregados e geralmente, a unidade de planejamento utilizada, é o menor segmento de vôo, que leva de uma cidade à outra, sem paradas intermediárias ou então uma rotação. Um passo importante dentro da geração da matriz, consiste em fazer a correspondência entre a lista completa das aeronaves (unidades de produção) e a lista dos segmentos de vôos. Isto é feito geralmente por uma rotina de atribuição. Uma operação bastante complicada é o teste para verificar se uma dada rotação é possível. Dado que estamos tratando com um problema combinatório, em número, as rotações geradas são muitas e além da necessidade de aproveitar somente as possíveis, buscamos também a conveniência de reduzí-las a uma quantidade manuseável. Naturalmente, após a obtenção da solução matemática para o problema, surge a questão, já fora da fase de otimização, de fazer a ligação dos vôos em rotações, utilizando a "escalação de serviço" que determina os membros das tripulações. Resumindo, o gerador desempenha a tarefa de produzir toda a seqüência combinatória de rotações e o nosso algoritmo faz a escolha ótima ou ainda, escolhas viáveis das rotações.

6.4 - A QUESTÃO DOS CUSTOS.

Os custos de cada rotação são formados, dentre outras, com os salários pagos aos componentes das tripulações, prêmios, seguros, estadias, manutenção, serviço de bordo, etc. Algumas empresas pagam seus pilotos um sa

lário fixo, sem levar em conta o desempenho pessoal ou a quantidade de trabalho. Outras já adotam um esquema de remuneração diferente e pagam proporcional ao desempenho de cada um. São adotados, de um modo geral, desde os mais simples critérios aos mais complexos, na determinação da forma de remuneração. Naturalmente, do ponto de vista de quem modela, este aspecto tem pouca importância.

6.5 - AS REDUÇÕES.

As reduções propostas no CAPÍTULO III, para supressão de linhas e colunas de uma matriz 0-1, no caso de modelos de recobrimento e particionamento, são naturalmente aplicáveis aqui para a redução do tamanho da matriz de catalogação das rotações e se juntam àquelas reduções processadas pelo gerador após sua construção. Em problemas de grande tamanho, estas reduções são de grande valia, dado que o processo enumerativo das pseudo-soluções parciais fica naturalmente amenizado.

6.6 - ATRIBUIÇÃO (ALOCAÇÃO) INDIVIDUAL DE TRIPULANTES ÀS TRIPULAÇÕES.

Um outro problema cuja solução interessa às empresas aéreas, é o da alocação ótima de tripulantes às tripulações, principalmente no caso das tripulações não serem fixas (i. e., de serem variáveis de vôo para vôo). A questão surge quando se procura saber como catalogar as tripulações de modo a aproveitar a mobilidade dos tripulantes que chegam de diversas procedências e partem para vários destinos, levando em conta que diversos tipos de aeronaves são utilizados com a restrição de que certos tripulantes não podem manusear certos tipos de aviões ou não estão autorizados para tanto (por questão de treinamento, saúde, condições contratuais, etc).

Para ilustrar, imaginemos o caso de dois vôos com procedências distintas, chegando quase simultaneamente numa mesma cidade. Após a chegada seus comandantes devem fornecer relatórios dos respectivos vôos ao escritório da empresa e para isso dispõem de um certo período (por exemplo, 40 min). Além disso, dispõem de um tempo para descanso e alimentação antes de se dirigirem novamente à administração para receberem a designação para um novo vôo, relatórios das condições do tempo e do tráfego aéreo. Neste ponto, os tripulantes chegados de procedências distintas, são candidatos aparentemente em iguais condições, ao próximo vôo. Entretanto, surge uma série de conveniências para a designação de determinados tripulantes para o vôo seguinte. Estas conveniências atendem muitas vezes às restrições impostas ao trabalho dos tripulantes (número máximo de horas de vôo por período ou por unidade de planejamento, número máximo de dias fora da base, residência, etc). Mais uma vez surge a possibilidade de se resolver este problema com recursos combinatórios, bastando para tanto a geração da matriz catálogo, onde nas colunas temos as rotações possíveis para os tripulantes e nas linhas os vôos disponíveis. O modelo seria também formado por (VI.1), (VI.2), (VI.3) e (VI.4). Se tomarmos os custos todos unitários ($c_j = 1, \forall j \in N$), estaremos minimizando o número de determinado membro de tripulação, caso contrário minimizamos o custo total das rotações. A restrição ($\geqslant$), permite aos tripulantes viajarem como passageiros para assumir um próximo vôo.

6.7 - UM TRATAMENTO ALTERNATIVO DA MODELAGEM.

Imaginemos que a matriz catálogo seja gerada, de tal forma que grupos de colunas correspondam às combinações possíveis de vôos, formando rotações manipuláveis por uma tripulação. Assim, as entradas da matriz serão: $a_{ij(k)} = 1$, se o i-ésimo vôo pode ser feito pela k-ésima combinação atribuida à j-ésima tripulação e $a_{ij(k)} = 0$, caso contrário. Na solução, $x_{j(k)} = 1$ se a k-ésima combinação foi escolhida para a j-ésima tripulação e $x_{j(k)} = 0$

caso contrário. Posto que cada vôo deve ser executado por uma única (ou pelo menos por uma) tripulação e que $c_{j(k)}$ representa o custo da k-ésima combinação atribuida à j-ésima tripulação, o modelo agora tem a forma:

$$\text{minimizar } z = \sum_j \sum_k c_{j(k)} x_{j(k)} \tag{VI.5}$$

$$\text{sujeito a} \quad \sum_j \sum_k a_{ij(k)} x_{j(k)} = (\geqslant) 1 \tag{VI.6}$$

$$a_{ij(k)} \in \{0,1\} \tag{VI.7}$$

$$x_{j(k)} \in \{0,1\} \tag{VI.8}$$

Observemos que neste modelo temos naturalmente a restrição de que cada tripulação seja alocada a somente uma combinação (rotação). Isto é traduzido por:

$$\sum_k x_{j(k)} \leqslant 1 \tag{VI.9}$$

Uma outra restrição pode impor que uma única tripulação (ou no máximo um certo número de tripulações) de cada base seja utilizada. Esta será uma restrição adicional denominada "restrição da base". A figura VI.1, mostra o aspecto da matriz catálogo do modelo contendo também as restrições adicionais que acabamos de mencionar.

6.8 - UMA ILUSTRAÇÃO NUMÉRICA.

A solução de qualquer problema de catalogação e alocação de tripulações, de interesse prático, envolveria a geração da matriz das restri

TRIPULAÇÃO(J)	1			2		3		...
COMBINAÇÃO(K)	1	2	3	1	2	1	2	...
BASE DA TRIPULAÇÃO	1			2		2		...
VARIÁVEL $X_{J(K)}$	$X_{1(1)}$	$X_{1(2)}$	$X_{1(3)}$	$X_{2(1)}$	$X_{2(2)}$	$X_{3(1)}$	$X_{3(2)}$	...
VÔO I								
1	1	0	0	1	0	.....		... ≥ 1
2	0	0	1	0	1	.....		... ≥ 1
3	1	1	1	0	1	.....		... ≥ 1
⋮	⋮	⋮	⋮	⋮	⋮	⋮⋮⋮⋮⋮		⋮⋮⋮
m	1	1	0	0	1	.....		... ≥ 1
No máximo uma combinação po de ser usada.	1	1	1					≤ 1
				1	1			≤ 1
						1	1	≤ 1
Restrições im postas sobre as bases de tripulações.	1	1	1					≤ 1
				1	1	1	1	≤ 1

Figura VI.1

Matriz catálogo com as restrições adicionais.

ções, matriz catálogo, fato que foge aos objetivos deste trabalho. A intenção deste capítulo, é meramente indicar como o código que desenvolvemos pode ajudar um especialista de Pesquisa Operacional, na solução de inúmeros problemas combinatórios que aceitassem uma modelagem semelhante a que tem sido objeto de nossa atenção. Naturalmente, após a geração da matriz catálogo, além de pequenas modificações possivelmente para incluir aspectos particulares do problema, como restrições distintas das habituais, o programa desenvolvido estaria apto a buscar a solução do problema.

Vejamos agora um simples exemplo numérico, para ilustrar. Imaginemos que seja fornecida uma tabela onde para cada vôo temos as seguintes informações:

Vôo k

número do vôo NV(k)	data do vôo DV(k)	local de chegada LC(k)	local de saída LS(k)	duração do vôo DUR(k)	custo do vôo C(k)

Vejamos o aspecto das restrições para a geração da matriz catálogo das rotações. Seja a combinação (k1,k2,k3). Se ela é uma rotação válida, temos:

1. $LC(k1) = LS(k2)$ (VI.10)

2. $LC(k2) = LS(k3)$ (VI.11)

3. $HS(k2) - HC(k1) \geqslant p$ (minutos ou horas) (VI.12)

4. $HS(k3) - HC(k2) \geqslant p$ " " (VI.13)

5. $DUR(k1) + DUR(k2) + DUR(k3) \leqslant q$ (horas) (VI.14)

6. $\text{card}\left\{k1,k2,k3\right\} \leqslant r$ (r é o número máximo de vôos para uma dada tripulação num período). (VI.15)

7. $DV(k3) - DV(k1) \leqslant s$ (s é o número de dias que uma tripulação pode ficar fora de sua base). (VI.16)

O significado das restrições é facilmente compreensível. A restrição (VI.10) junto com a (VI.11), impõem que o local de chegada de um vôo deva coincidir com o de saída do seguinte. (VI.12) e (VI.13), impõem que entre dois vôos consecutivos, as tripulações devam dispor de um certo tempo para troca de relatórios, repouso, alimentação, etc. (VI.14) se refere a um máximo de horas de vôo por período.

Na figura VI.2, temos representados pelos traços, alguns vôos entre algumas cidades (ida e/ou volta). Fizemos uma geração manual de 75 rotações viáveis (combinações possíveis) com os 16 vôos. Assim, um modelo de 75 variáveis e 16 restrições foi constuído, simplesmente para fins de ilustração e teste do programa. Foi minimizado o custo total de operação, como também minimizado o número de tripulações necessárias para manipular os vôos. A despreocupação com que foi feita esta ilustração numérica, certamente nos impede de fazer maiores comentários sobre os resultados (que foram coerentes), a não ser o tempo de processamento surpreendente para a solução destes problemas.

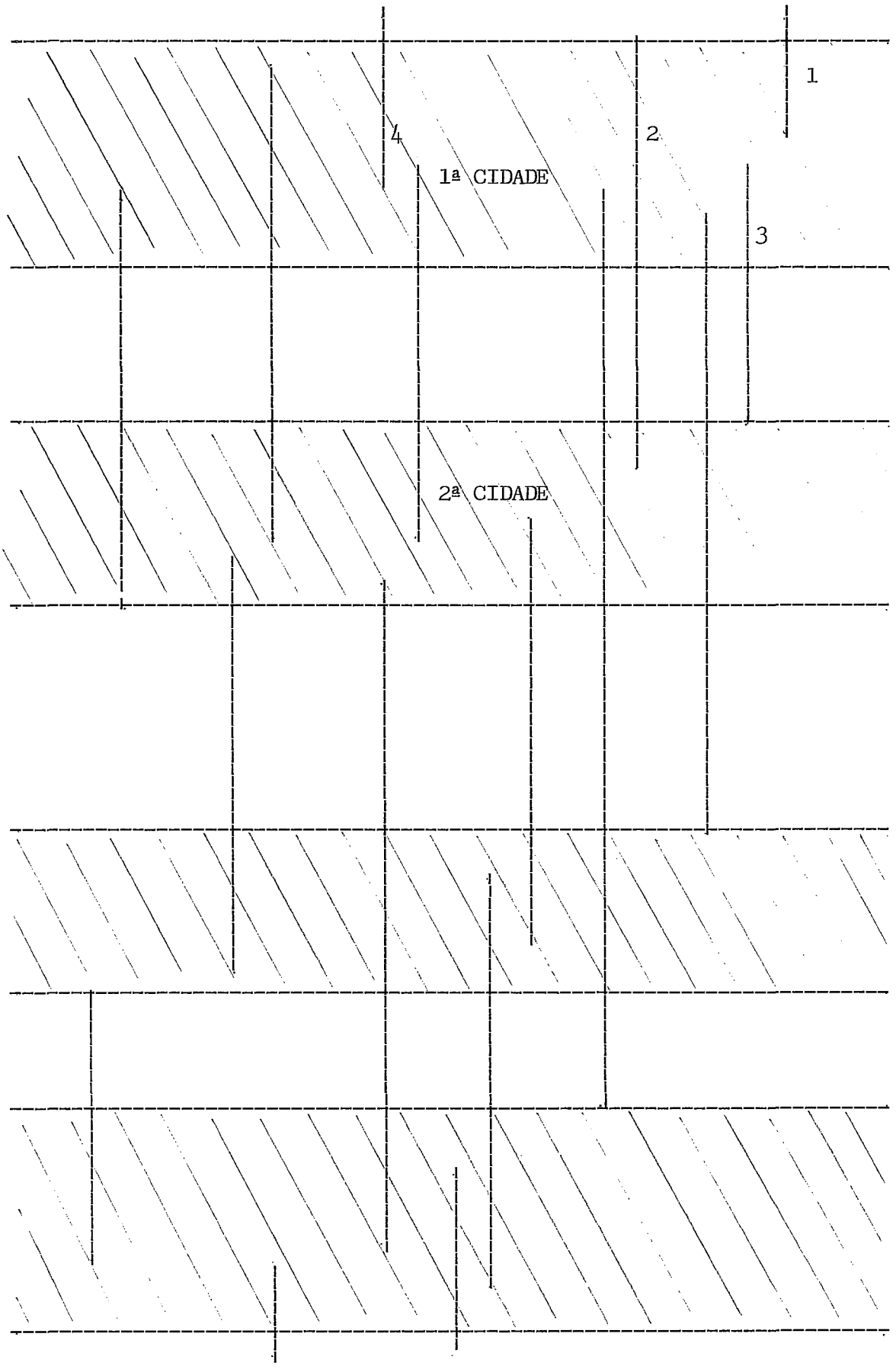


Figura VI.2. Um esquema simples para catalogar vôos.

APÊNDICE

a. Programa em ALGOL/B6700, para:

i. Solução de problemas de Programação Inteira bivalente.

ii. Determinação dos conjuntos recobrimento e particionamento mínimos.

iii. Determinação do emparelhamento máximo e recobrimento mínimo em grafos não orientados.

b. Dados utilizados para teste do programa.

```
BEGIN
   FILE ENTRADA(KIND=READER), SAIDA(KIND=PRINTER);
   INTEGER M,MR,N;
%-------------------------------------------------------------------%
   PROCEDURE DETXMIN(APL,NR,DECOL,DELIN,COLPIV,IX,MIN);
           VALUE NR,DECOL,DELIN;
           INTEGER NR,COLPIV,DECOL,DELIN;
           ARRAY APL[0];
           INTEGER ARRAY IX[0];
           REAL MIN;
%---------DETXMIN DETERMINA O MINIMO DOS NUMEROS
%---------APL[NR*DECOL+IX[K]*DELIN], K=1,...,IX[0].
           BEGIN
              INTEGER K;
              COLPIV:=IX[1];
              MIN:=APL[NR*DECOL+IX[1]*DELIN];
              FOR K:=2 STEP 1 UNTIL IX[0]
              DO
              IF APL[NR*DECOL+IX[K]*DELIN] < MIN
              THEN
                BEGIN
                   MIN:=APL[NR*DECOL+IX[K]*DELIN];
                   COLPIV:=IX[K]
                END
           END;
%-------------------------------------------------------------------%
   PROCEDURE PROXLIN(APL,DECOL,DELIN,INDVB,INDVN,M,N,TP,ZA,
                     ZXMENOR);
           VALUE DECOL,DELIN,M,N;
           ARRAY APL[0];
           INTEGER DECOL,DELIN,M,N,TP,ZA,ZXMENOR;
           INTEGER ARRAY INDVN[1],INDVB[1];
%---------ROTINA PARA CALCULAR O MINIMO DE UM PROBLEMA DE PROGRA-
%---------MACAO LINEAR PELO METODO SIMPLEX REVISADO. NESTA FORMA
%---------O METODO ASSUME QUE A ORIGEM E UMA SOLUCAO VIAVEL.   O
%---------METODO USA A PROCEDURE GLOBAL DETXMIN.
```

```
        BEGIN
            INTEGER I,M,LINPIV,COLPIV,DECOL1,DELIN1,S,J;
            REAL QX,MIN,EPSILON;
            BOOLEAN COMPLETOU,SALTA;
            INTEGER ARRAY IX[0:N],IY[0:M],IZ[0:M];
            ARRAY BPL[0:M*(N+2)],CPL[0:M];
%-----------------------------------------------------------%
            PROCEDURE INVERTE(BK,CK,M,LINPIV,S,DECOL2,DELIN2);
                    VALUE M,S,LINPIV,DECOL2,DELIN2;
                    INTEGER M,S,LINPIV,DECOL2,DELIN2;
                    ARRAY BK[0],CK[0];
%--------------------ROTINA PARA CALCULAR PASSO A PASSO A INVERSA
%--------------------DA MATRIZ QUE OCORRE NO SIMPLEX REVISADO.
                    BEGIN
                       INTEGER I,K;
                       FOR K:=1 STEP 1 UNTIL S
                       DO
                       IF BK[LINPIV*DECOL2+K*DELIN2] ¬= 0
                       THEN
                         FOR I:=0 STEP 1 UNTIL LINPIV-1,LINPIV+1 STEP 1
                                         UNTIL M
                         DO
                         IF CK[I] ¬= 0
                         THEN
                           BK[I*DECOL2+K*DELIN2]:=BK[I*DECOL2+K*DELIN2]+
                              BK[LINPIV*DECOL2+K*DELIN2]*CK[I];
                       FOR K:=1 STEP 1 UNTIL S
                       DO
                       BK[LINPIV*DECOL2+K*DELIN2]:=BK[LINPIV*DECOL2+
                          K*DELIN2]*CK[LINPIV]
                    END;
%-----------------------------------------------------------%
            FOR K:=1 STEP 1 UNTIL N
            DO
            INDVN[K]:=IX[K]:=K;
            FOR I:=1 STEP 1 UNTIL M
```

```
            DO
            INDVB[I]:=N+I;
            IX[0]:=N;
            IY[0]:=0;
            DECOL1:=1;
            DELIN1:=M+1;
            EPSILON:=2**(-30);
%-----------GERACAO DA MATRIZ IDENTIDADE.
            FOR I:=0 STEP 1 UNTIL M
            DO
            FOR K:=1 STEP 1 UNTIL M
            DO
            BPL[I*DECOL1+K*DELIN1]:=IF K = I THEN -1 ELSE 0;
            COMPLETOU:=FALSE;
            WHILE ¬ COMPLETOU AND APL[0]+ZXMENOR-1-ZA >= 0
            DO
            BEGIN
               SALTA:=FALSE;
%--------------DETERMINACAO DA COLUNA PIVO
               QX:=MIN:=0;
               IF IX[0] ¬= 0
               THEN
                 DETXMIN(APL,0,DECOL,DELIN,COLPIV,IX,MIN);
               IF IY[0] ¬= 0
               THEN
                 FOR I:=1 STEP 1 UNTIL IY[0]
                 DO
                 IF BPL[0*DECOL1+IY[I]*DELIN1] < QX
                 THEN
                   BEGIN
                      S:=I;
                      QX:=BPL[0*DECOL1+IY[I]*DELIN1]
                   END;
               IF MIN >= -EPSILON AND QX >= -EPSILON
               THEN
                 BEGIN
```

```
                TP:=0;
                COMPLETOL:=TRUE
              END
            ELSE
              BEGIN
                INTEGER CONT;
                BOOLEAN ENCONTROU;
                K:=0;
                IF MIN > QX
                THEN
                  BEGIN
                    K:=1;
                    COLPIV:=IZ[S];
                    FOR I:=0 STEP 1 UNTIL M
                    DO
                    CPL[I]:=-BPL[I*DECOL1+IY[S]*DELIN1];
                  END
                ELSE
                  BEGIN
                    CPL[0]:=-APL[0*DECOL+COLPIV*DELIN];
                    FOR I:=1 STEP 1 UNTIL M
                    DO
                    BEGIN
                      QX:=0;
                      FOR J:=1 STEP 1 UNTIL M
                      DO
                      IF APL[J*DECOL+COLPIV*DELIN] ¬= 0  AND
                         BPL[I*DECOL1+J*DELIN1] ¬= 0
                      THEN
                        QX:= * + APL[J*DECOL+COLPIV*DELIN]*
                                 BPL[I*DECOL1+J*DELIN1];
                      CPL[I]:=QX
                    END
                  END;
%-------------------DETERMINACAO DA LINHA PIVO.
                LINPIV:=0;
```

```
                CONT:=0;
                ENCONTROU:=FALSE;
                FOR I:=1 STEP 1 WHILE I <= M AND ¬ ENCONTROU
                DO
                IF CPL[I] <= 0
                THEN
                  CONT:= * + 1
                ELSE
                  BEGIN
                     ENCONTROU:=TRUE;
                     QX:=APL[I*DECOL]/CPL[I];
                     LINPIV:=I;
                     J:=LINPIV;
                     FOR I:=J+1 STEP 1 UNTIL M
                     DO
                     IF CPL[I] > 0
                     THEN
                       BEGIN
                          IF APL[I*DECOL]/CPL[I] < QX
                          THEN
                            BEGIN
                               QX:=APL[I*DECOL]/CPL[I];
                               LINPIV:=I
                            END
                          ELSE
%-------------------------------MANIPULACAO DE POSSIVEIS DEGENERACOES*
                            IF APL[I*DECOL]/CPL[I] = QX
                            THEN
                              BEGIN
                                 INTEGER K;
                                 REAL QX1,QX2;
                                 K:=0;
                                 DO
                                 BEGIN
                                    K:= * + 1;
                                    QX1:=APL[LINPIV*DECOL+K*DELIN]/
```

```
                                        CPL[LINPIV];
                                    QX2:=APL[I*DECOL+K*DELIN]/
                                        CPL[I]
                                END
                                UNTIL QX1 ¬= QX2 OR K = N;
                                IF QX1 > QX2
                                THEN
                                  LINPIV:=I
                            END
                    END;
                    FOR I:=0 STEP 1 UNTIL LINPIV-1,LINPIV+1 STEP 1
                                    UNTIL M
                    DO
                    CPL[I]:=-CPL[I]/CPL[LINPIV];
                    CPL[LINPIV]:=1/CPL[LINPIV];
%-------------------------TRANSFORMACAO DA PRIMEIRA COLUNA DO TABLEAU
%-------------------------APL.
                    INVERTE(APL,CPL,M,LINPIV,1,DECOL,0);
%-------------------------TRANSFORMACAO DA PRIMEIRA LINHA DO TABLEAU
%-------------------------APL.
                    FOR J:=1 STEP 1 UNTIL COLPIV-1,COLPIV+1 STEP 1
                                    UNTIL N
                    DO
                    BEGIN
                       QX:=0;
                       FOR I:=1 STEP 1 UNTIL M
                       DO
                       IF BPL[LINPIV*DECOL1+I*DELIN1] ¬= 0
                       THEN
                         QX:= * + BPL[LINPIV*DECOL1+I*DELIN1]*
                                  APL[I*DECOL+J*DELIN];
                       APL[0*DECOL+J*DELIN]:=APL[0*DECOL+J*DELIN]-
                                             QX*CPL[0]
                    END;
                    APL[0*DECOL+COLPIV*DELIN]:=-CPL[0];
%-------------------------TRANSFORMACAO DA INVERSA.
```

```
INVERTE(BPL,CPL,M,LINPIV,M,DECOL1,DELIN1);
IF K = 0
THEN
  BEGIN
     FOR J:=1 STEP 1 WHILE J <= IY[0] AND
                              ¬ SALTA
     DO
     IF LINPIV = IY[J]
     THEN
       BEGIN
          IF IZ[J] = COLPIV
          THEN
            BEGIN
               K:=INDVN[COLPIV];
               INDVN[COLPIV]:=INDVB[LINPIV];
               INDVB[LINPIV]:=K;
               SALTA:=TRUE
            END
          ELSE
            BEGIN
               FOR I:=1 STEP 1 UNTIL IX[0]
               DO
               IF IX[I] = COLPIV
               THEN
                 IX[I]:=IZ[J];
               IZ[J]:=COLPIV;
               K:=INDVN[COLPIV];
               INDVN[COLPIV]:=INDVB[LINPIV];
               INDVB[LINPIV]:=K;
               SALTA:=TRUE
            END
       END;
     IF ¬ SALTA
     THEN
       BEGIN
          IY[0]:=IY[0]+1;
```

```
            IY[IY[0]]:=LINPIV;
            IZ[IY[0]]:=COLPIV;
            FOR I:=1 STEP 1 WHILE I<= IX[0] AND
                                   ¬ SALTA
            DO
            IF IX[I] = COLPIV
            THEN
              BEGIN
                 IX[0]:=IX[0]-1;
                 FOR J:=I STEP 1 UNTIL IX[0]
                 DO
                 IX[J]:=IX[J+1];
                 K:=INDVN[COLPIV];
                 INDVN[COLPIV]:=INDVB[LINPIV];
                 INDVB[LINPIV]:=K;
                 SALTA:=TRUE
              END
         END
    END;
IF ¬ SALTA
THEN
  BEGIN
     FOR J:=1 STEP 1 WHILE J <= IY[0] AND
                           ¬ SALTA
     DO
     IF LINPIV = IY[J]
     THEN
       BEGIN
          K:=INDVN[COLPIV];
          INDVN[COLPIV]:=INDVB[LINPIV];
          INDVB[LINPIV]:=K;
          SALTA:=TRUE
       END;
     IF ¬ SALTA
     THEN
       BEGIN
```

```
                                    IX[0]:=IX[0]+1;
                                    IX[IX[0]]:=IZ[S];
                                    IY[C]:=IY[0]-1;
                                    FOR J:=S STEP 1 UNTIL IY[0]
                                    DO
                                    BEGIN
                                       IY[J]:=IY[J+1];
                                       IZ[J]:=IZ[J+1]
                                    END;
                                    K:=INDVN[COLPIV];
                                    INDVN[COLPIV]:=INDVB[LINPIV];
                                    INDVB[LINPIV]:=K
                                 END
                              END
                           END;
                        IF CONT = M
                        THEN
                           BEGIN
                              TP:=1;
                              COMPLETOU:=TRUE
                           END
                     END
                  END
               END;
%-------------------------------------------------------------------%
   PROCEDURE SOLXDUAL(APL,AVANCA,DECOL,DELIN,GERA,JJ,U,ZA,
                      ZXMENOR);
         VALUE JJ;
         ARRAY APL[0],U[1];
         INTEGER AVANCA,DECOL,DELIN,JJ,ZA,ZXMENOR;
         BOOLEAN GERA;
         BEGIN
            INTEGER I,MA,NA,IP;
            INTEGER ARRAY INDVA[1:(M+JJ)],INDVB[1:JJ];
            MA:=M;
            NA:=N;
```

```
            N:=M+JJ;
            M:=JJ;
            PRCXLIN(APL,DECCL,CELIN,INDVB,INDVN,M,N,TP,ZA,
                    ZXMENOR);
            AVANCA:=APL[0]+ZXMENOR-1-ZA;
            IF TP = 0 AND AVANCA >= 0
            THEN
              BEGIN
                 FOR I:=1 STEP 1 UNTIL M
                 DO
                 IF INDVB[I] <= MA
                 THEN
                   U[INDVB[I]]:=APL[I*DECOL]
              END
            ELSE
              GERA:=FALSE;
            M:=MA;
            N:=NA
         END;
%-------------------------------------------------------------%
   PROCEDURE RESXSUB(A,AVANCA,B,C,DESTVOAS,DISPONIVEL,INDICE,NO,
                     PRIXLIN,PRIXCOL,TERRA,W,X,ZA,ZERO,ZXMENOR);
         INTEGER AVANCA,DISPONIVEL,NO,TERRA,ZA,ZXMENOR;
         INTEGER ARRAY A[0,1],B[1],C[1],DESTVOAS[1],INDICE[1],
                       PRIXLIN[1],PRIXCOL[1],W[1],X[1],ZERO[1];
         BEGIN
            REAL CONS;
            ARRAY U[1:M];
            INTEGER DECOL,DELIN,I,J,JK,JF,KK,LD,LM;
            BOOLEAN GERA;
%-------------------------------------------------------------%
            GERA:=TRUE;
            KK:=ZA:=0;
            FOR J:=1 STEP 1 UNTIL N
            DO
            BEGIN
```

```
    ZA:=ZA+X[J]*C[J];
    IF X[J] = 0 AND ZERO[J] ¬= 0
                AND INDICE[J] = J
                AND PRIXCOL[J] ¬= TERRA
    THEN
      KK:= * + 1
END;
IF KK > 0
THEN
  BEGIN
     ARRAY APL[0:(M+KK+1)*(KK+1)-1],NB[1:M];
     DECOL:=M+KK+1;
     DELIN:=1;
     APL[0]:=0;
     LD:=0;
     FOR I:=1 STEP 1 UNTIL M
     DO
     BEGIN
        JK:=PRIXLIN[I];
        IF JK ¬= TERRA
        THEN
          BEGIN
             LD:= * + 1;
             APL[LD]:=-B[A[1,JK]]
          END
     END;
     JP:=M;
     M:=LD;
     FOR J:=1 STEP 1 UNTIL KK
     DO
     FOR I:=1 STEP 1 UNTIL M+KK
     DO
     APL[J*DECOL+I*DELIN]:=0;
     KK:=0;
     FOR J:=1 STEP 1 UNTIL N
     DO
```

```
                IF X[J] = 0 AND ZERO[J] ¬= 0
                            AND INDICE[J] = J
                            AND PRIXCOL[J] ¬= TERRA
                THEN
                  BEGIN
                     KK:= * + 1;
                     APL[KK*DECCL]:=C[J];
                     JK:=PRIXCOL[KK];
                     WHILE JK ¬= TERRA
                     DO
                     BEGIN
                        APL[KK*DECCL+(A[1,JK]-DESTVOAS[A[1,JK]])
                            *DELIN]:=-A[0,JK];
                        JK:=A[3,JK]
                     END;
                  END;
                FOR J:=1 STEP 1 UNTIL N
                DO
                IF X[J] = 1
                THEN
                  BEGIN
                     JK:=PRIXCOL[J];
                     WHILE JK ¬= TERRA
                     DO
                     BEGIN
                        APL[A[1,JK]]:=APL[A[1,JK]]+A[0,JK];
                        JK:=A[3,JK]
                     END
                  END;
                FOR I:=1 STEP 1 UNTIL M
                DO
                NB[I]:=APL[I];
%---------------GERACAO DAS RESTRICOES 0-1.
                FOR J:=1 STEP 1 UNTIL KK
                DO
                APL[J*DECCL+(M+J)*DELIN]:=APL[M+J]:=1;
```

```
            SOLXDUAL(APL,AVANCA,DECCL,DELIN,GERA,KK,U,ZA,
                     ZXMENOR);
            IF GERA
            THEN
              BEGIN
                 CONS:=0;
                 FOR I:=1 STEP 1 UNTIL M
                 DO
                 CONS:= * + U[I]*NB[I];
                 CONS:= * + ZXMENOR=1=ZA;
                 M:=JP;
                 LM:=0;
                 FOR I:=1 STEP 1 UNTIL M
                 DO
                 BEGIN
                    JK:=PRIXLIN[I];
                    IF JK ¬= TERRA
                    THEN
                      BEGIN
                         LM:= * + 1;
                         U[A[1,JK]]:=U[LM]
                      END
                 END
              END;
%-----------GERACAO DAS RESTRICOES SUBSTITUTAS.
          IF GERA AND KK > 0
          THEN
            BEGIN
               ARRAY AS[1:4,1:N],BS[1:4],COEF[1:N];
               FOR J:=1 STEP 1 UNTIL N
               DO
               IF X[J] = 0 AND ZERO[J] ¬= 0
                           AND INDICE[J] = J
               THEN
                 BEGIN
```

```
                    COEF[J]:=0;
                    JK:=PRIXCOL[J];
                    WHILE JK <> TERRA
                    DO
                    BEGIN
                       COEF[J]:=COEF[J]+L[A[1,JK]]*A[0,JK];
                       JK:=A[3,JK]
                    END;
                    COEF[J]:=COEF[J]-C[J]
                  END
                ELSE
                  COEF[J]:=0;
%-----------------COLOCACAO DAS RESTRICOES SUBSTITUTAS
%-----------------NA AREA RESERVADA PARA ELAS.
                FOR I:=4 STEP -1 UNTIL 2
                DO
                BEGIN
                   BS[I]:=BS[I-1];
                   FOR J:=1 STEP 1 UNTIL N
                   DO
                   AS[I,J]:=AS[I-1,J]
                END;
                FOR J:=1 STEP 1 UNTIL N
                DO
                AS[1,J]:=COEF[J];
                BS[1]:=CONS;
%-----------------TESTA COM BASE NAS RESTRICOES SUBSTITUTAS
%-----------------QUAIS AS VARIAVEIS QUE DEVEM SER MANTIDAS
%-----------------NO NIVEL ZERO E QUAIS AS QUE DEVEM PASSAR
%-----------------AO NIVEL 1.
                FOR I:=1 STEP 1 UNTIL 4
                DO
                BEGIN
                   INTEGER SOMA;
                   SOMA:=0;
                   FOR J:=1 STEP 1 UNTIL N
```

```
DO
IF AS[I,J] > 0
THEN
  SOMA:= * + AS[I,J];
SOMA:= * + BS[I];
IF SOMA >= 0
THEN
  BEGIN
     FOR J:=1 STEP 1 UNTIL N
     DO
     IF SOMA-ABS(AS[I,J]) < 0
     THEN
       BEGIN
          IF AS[I,J] < 0 AND X[J] = 0
                         AND INDICE[J] =
                             J
                         AND ZERO[J] ¬= 0
          THEN
            BEGIN
               NO:=DISPONIVEL;
               DISPONIVEL:= * + 1;
               W[NO]:=J;
               X[J]:=0;
               ZERO[J]:=0
            END;
          IF AS[I,J] > 0 AND X[J] = 0
                         AND INDICE[J] =
                             J
                         AND ZERO[J] ¬= 0
          THEN
            BEGIN
               NO:=DISPONIVEL;
               DISPONIVEL:= * + 1;
               W[NO]:=J;
               X[J]:=1
            END
```

```
                        END
                    END
                END
            END
        END;
%-------------------------------------------------------------%
    PROCEDURE RESTRICXS(A,AVANCA,B,C,DISPONIVEL,INDICE,NO,W,X,
                        ZA,ZERO,ZXMENOR);
        INTEGER AVANCA,DISPONIVEL,NO,ZA,ZXMENOR;
        INTEGER ARRAY A[1,1],B[1],C[1],INDICE[1],W[1],X[1],
                      ZERO[1];
        BEGIN
            REAL CONS;
            ARRAY U[1:M];
            INTEGER I,J,JJ,DECOL,DELIN;
            BOOLEAN GERA;
%-------------------------------------------------------------%
            GERA:=TRUE;
            JJ:=ZA:=0;
            FOR J:=1 STEP 1 UNTIL N
            DO
            BEGIN
                ZA:=ZA+X[J]*C[J];
                IF X[J] = 0 AND ZERO[J] =/= 0
                            AND INDICE[J] = J
                THEN
                  JJ:= * + 1
            END;
            IF JJ > 0
            THEN
              BEGIN
                ARRAY APL[0:(M+JJ+1)*(JJ+1)-1],NB[1:M];
                DECOL:=M+JJ+1;
                DELIN:=1;
                APL[0]:=0;
                FOR I:=1 STEP 1 UNTIL M
```

```
                DO
                APL[I]:=B[I];
                FOR J:=1 STEP 1 UNTIL JJ
                DO
                FOR I:=1 STEP 1 UNTIL M+JJ
                DO
                APL[J*DECOL+I*DELIN]:=0;
                JJ:=0;
                FOR J:=1 STEP 1 UNTIL N
                DO
                IF X[J] = 0 AND ZERO[J] ¬= 0
                            AND INDICE[J] = J
                THEN
                  BEGIN
                     JJ:= * + 1;
                     APL[JJ*DECOL]:=C[J];
                     FOR I:=1 STEP 1 UNTIL M
                     DO
                     APL[JJ*DECOL+I*DELIN]:=-A[I,J]
                  END;
                FOR J:=1 STEP 1 UNTIL N
                DO
                IF X[J] = 1
                THEN
                  FOR I:=1 STEP 1 UNTIL M
                  DO
                  APL[I]:=APL[I]+A[I,J];
                FOR I:=1 STEP 1 UNTIL M
                DO
                NB[I]:=APL[I];
%---------------GERACAO DAS RESTRICOES 0-1.
                FOR J:=1 STEP 1 UNTIL JJ
                DO
                APL[J*DECOL+(M+J)*DELIN]:=APL[M+J]:=1;
                SOLXDUAL(APL,AVANCA,DECCL,DELIN,GERA,JJ,U,ZA,
                         ZXMENOR);
```

```
                    IF GERA
                    THEN
                      BEGIN
                         CONS:=0;
                         FOR I:=1 STEP 1 UNTIL M
                         DO
                         CONS:= * + U[I]*NB[I];
                         CONS:= * + ZXMENOR-1-ZA
                      END
                  END;
%-------------GERACAO DAS RESTRICOES SUBSTITUTAS.
                IF GERA AND JJ > 0
                THEN
                  BEGIN
                     ARRAY AS[1:2,1:N],BS[1:2],COEF[1:N];
                     FOR J:=1 STEP 1 UNTIL N
                     DO
                     IF X[J] = 0 AND ZERO[J] =/= 0
                                 AND INDICE[J] = J
                     THEN
                       BEGIN
                          COEF[J]:=0;
                          FOR I:=1 STEP 1 UNTIL M
                          DO
                          COEF[J]:=COEF[J]+U[I]*A[I,J];
                          COEF[J]:=COEF[J]-C[J]
                       END
                     ELSE
                       COEF[J]:=0;
%-----------------COLOCACAO DAS RESTRICCES SUBSTITUTAS NA AREA RESERVADA
%-----------------PARA ELAS.
                     BS[2]:=BS[1];
                     FOR J:=1 STEP 1 UNTIL N
                     DO
                     BEGIN
                        AS[2,J]:=AS[1,J];
```

```
                AS[1,J]:=COEF[J]
              END;
              BS[1]:=CONS;
%-----------------TESTA COM BASE NAS RESTRICOES SUBSTITUTAS QUAIS AS
%-----------------VARIAVEIS QUE DEVEM SER MANTIDAS NO NIVEL ZERO E QUAIS
%-----------------AS QUE PODEM PASSAR AO NIVEL 1.
              FOR I:=1 STEP 1 UNTIL 2
              DO
              BEGIN
                INTEGER SOMA;
                SOMA:=0;
                FOR J:=1 STEP 1 UNTIL N
                DO
                IF AS[I,J] > 0
                THEN
                  SOMA:= * + AS[I,J];
                SOMA:= * + BS[I];
                IF SOMA >= 0
                THEN
                  BEGIN
                    FOR J:=1 STEP 1 UNTIL N
                    DO
                    IF SOMA-ABS(AS[I,J]) < 0
                    THEN
                      BEGIN
                        IF AS[I,J] < 0 AND X[J] = 0
                                       AND INDICE[J] = J
                                       AND ZERO[J] ¬= 0
                        THEN
                          BEGIN
                            NO:=DISPONIVEL;
                            DISPONIVEL:= * + 1;
                            W[NO]:=J;
                            X[J]:=0;
                            ZERO[J]:=0;
                          END;
```

```
                        IF A[I,J] > 0 AND X[J] = 0
                                      AND INDICE[J] = J
                                      AND ZERO[J] ¬= 0
                        THEN
                          BEGIN
                             NO:=DISPONIVEL;
                             DISPONIVEL:= * + 1;
                             W[NC]:=J;
                             X[J]:=1
                          END
                     END
                  END
               END
            END
         END;
%-----------------------------------------------------------------%
%------------INICIO DA ENUMERACAO IMPLICITA DAS PSEUDO-SOLUCOES
%------------PARCIAIS PARA OS PROBLEMAS DE DETERMINACAO     DOS
%------------CONJUNTOS RECOBRIMENTO E PARTICIONAMENTO MINIMOS,
%------------EMPARELHAMENTO MAXIMO  E RECOBRIMENTO MINIMO EM  UM
%------------GRAFO NAO ORIENTADO
%-----------------------------------------------------------------%
   PROCEDURE ENUXIMPXRECXPAR(A,B,C,INDICE,LINXSEG,PRIXLIN,
                             PRIXCOL,SOMA,TEMXSOLUCAO,TERRA,
                             X,XSOLXATUAL,Y,ZA,ZERO,ZXMENOR,
                             ZXOTIMO);
         VALUE TERRA;
         INTEGER TERRA,ZA,ZXMENOR,ZXOTIMO;
         INTEGER ARRAY A[0,1],B[1],C[1],INDICE[1],LINXSEG[0],
                       PRIXLIN[1],PRIXCOL[1],SOMA[1],X[1],
                       XSOLXATUAL[1],Y[1],ZERO[1];
         BOOLEAN TEMXSOLUCAO;
         BEGIN
            INTEGER AVANCA,DISPONIVEL,I,J,JO,L,LINHA,NC,
                    SUBLINHADO;
            INTEGER ARRAY DESTVDAS[1:M],V[1:N],W[1:N];
```

```
        BOOLEAN BUSCOUXDISP,ESGOTOUXW,PERMXLACO,VIAVEL;
%-------------------------------------------------------------%
        PROCEDURE SUBLIXCOMPL;
              BEGIN
                 INTEGER J;
                 J:=W[NO];
                 WHILE INDICE[J] = SUBLINHADO AND
                                   ¬ ESGOTOUXW
                 DO
                 BEGIN
                    INDICE[J]:=J;
                    X[J]:=0;
%-------------------------RETORNA O NO A PILHA DOS NOS
%-------------------------DISPONIVEIS
                    DISPONIVEL:=NO;
                    NO:= * - 1;
                    IF NO = 0
                    THEN
                      ESGOTOUXW:=TRUE
                    ELSE
                      J:=W[NO]
                 END;
                 IF ¬ ESGOTOUXW
                 THEN
                   BEGIN
                      X[J]:=1-X[J];
                      IF ZERO[J] = 0
                      THEN
                        ZERO[J]:=J;
                      INDICE[J]:=SUBLINHADO
                   END
              END;
%-------------------------------------------------------------%
        DISPONIVEL:=1;
        SUBLINHADO:=ZXMENOR:=5497558138887;
        BUSCOLXDISP:=TEMXSOLUCAO:=ESGOTOUXW:=FALSE;
```

```
        L:=0;
        DO
        BEGIN
           L:= * + 1;
           FOR I:=1 STEP 1 UNTIL L-1
           DO
           IF PRIXLIN[I] = TERRA
           THEN
             DESTVDAS[L]:=DESTVDAS[L]+1
        END
        UNTIL L = M;
        FOR J:=1 STEP 1 UNTIL N
        DO
        INDICE[J]:=ZERO[J]:=J;
        PERMXLACO:=TRUE;
        WHILE PERMXLACO
        DO
        BEGIN
%-----------CALCULO DO VETOR Y-----------------------------------------%
           INTEGER I;
           I:=0;
           DO
           BEGIN
              INTEGER KK,SOMA;
              LINHA:=LINXSEG[I];
              SOMA:=0;
              KK:=PRIXLIN[LINHA];
              DO
              BEGIN
                 SOMA:= * + A[0,KK]*X[A[2,KK]];
                 KK:=A[4,KK]
              END
              UNTIL KK = TERRA;
              Y[LINHA]:=SOMA-B[LINHA];
              I:= * + 1
           END
```

```
            UNTIL LINXSEG[I] = TERRA;
            I:=0;
            DO
            I:= * + 1
            UNTIL Y[I] < 0  OR LINXSEG[I] = TERRA;
            IF Y[I] >= 0
            THEN
              BEGIN
%----------------CALCULO DA FUNCAO OBJETIVO----------------------------%
                INTEGER J;
                ZA:=0;
                FOR J:=1 STEP 1 UNTIL N
                DO
                ZA:=ZA+C[J]*X[J];
                IF ZA < ZXMENOR
                THEN
                  BEGIN
                     TEMXSOLUCAO:=TRUE;
                     ZXMENOR:=ZA;
                     FOR J:=1 STEP 1 UNTIL N
                     DO
                     XSOLXATUAL[J]:=X[J];
                     WRITE(SAIDA,<//,X1,I6,/>,ZXMENOR);
                     WRITE(SAIDA,<X1,100I1>,FOR J:=1 STEP 1 UNTIL N DO
                                            X[J])
                  END;
                IF BUSCOUXDISP
                THEN
                  DO
                  BEGIN
                     SUBLIXCOMPL;
                     IF ESGOTOUXW
                     THEN
                       PERMXLACO:=FALSE
                     ELSE
                       RESXSUB(A,AVANCA,B,C,DESTVDAS,DISPONIVEL,
```

```
                                INDICE,NO,PRIXLIN,PRIXCOL,TERRA,
                                W,X,ZA,ZERO,ZXMENOR)
                    END
                    UNTIL AVANCA >= 0 OR ¬ PERMXLACO
                  ELSE
                    PERMXLACO:=FALSE
                END
              ELSE
                BEGIN
%-----------------CONSTRUCAO DO CONJUNTO V-------------------------------%
                  INTEGER I,J,K;
                  BOOLEAN ARRAY ENTROU[1:N];
                  FOR J:=1 STEP 1 UNTIL N
                  DO
                  ENTROU[J]:=FALSE;
                  ZA:=K:=L:=0;
                  FOR J:=1 STEP 1 UNTIL N
                  DO
                  ZA:=ZA+C[J]*X[J];
                  I:=0;
                  I:=LINXSEG[I];
                  DO
                  BEGIN
                     IF Y[I] < 0
                     THEN
                       BEGIN
                          INTEGER KK;
                          KK:=PRIXLIN[I];
                          DO
                          BEGIN
                             IF X[A[2,KK]] = 0                AND
                                INDICE[A[2,KK]] = A[2,KK]     AND
                                ¬ ENTROU[A[2,KK]]             AND
                                ZERO[A[2,KK]] = A[2,KK]
                             THEN
                               IF ZA+C[A[2,KK]] < ZXMENOR     AND
```

```
                    A[0,KK] > 0
                  THEN
                    BEGIN
                      K:= * + 1;
                      V[K]:=A[2,KK];
                      ENTROU[A[2,KK]]:=TRUE;
                      L:=K
                    END;
                  KK:=A[4,KK]
              END
              UNTIL KK = TERRA
          END;
          I:=LINXSEG[I]
      END
      UNTIL I = TERRA;
      IF L <> 0
      THEN
        BEGIN
%---------------------TESTA VIABILIDADE-------------------------------%
          INTEGER I,J,K;
          INTEGER ARRAY SOMA[1:M];
          VIAVEL:=TRUE;
          I:=0;
          DO
          BEGIN
            I:=LINXSEG[I];
            SOMA[I]:=0;
            IF Y[I] < 0
            THEN
              BEGIN
                INTEGER JK;
                JK:=PRIXLIN[I];
                DO
                BEGIN
                  IF ENTROU[A[2,JK]] AND
                    A[0,JK] > 0
```

```
                    THEN
                      SOMA[I]:=SOMA[I]+A[0,JK];
                    JK:=A[4,JK]
                  END
                  UNTIL JK = TERRA
                END;
              SOMA[I]:=SOMA[I]+Y[I]
            END
            UNTIL SOMA[I] < 0 OR LINXSEG[I] = TERRA;
            IF SOMA[I] < 0
            THEN
              VIAVEL:=FALSE;
            IF ¬ VIAVEL
            THEN
              IF BLSCOUXDISP
              THEN
                OC
                BEGIN
                   SUBLIXCOMPL;
                   IF ESGOTOUXW
                   THEN
                     PERMXLACO:=FALSE
                   ELSE
                     RESXSUB(A,AVANCA,B,C,DESTVDAS,
                             DISPONIVEL,INDICE,NC,
                             PRIXLIN,PRIXCOL,TERRA,
                             W,X,ZA,ZERO,ZXMENOR)
                END
                UNTIL AVANCA >= 0 OR ¬ PERMXLACO
              ELSE
                PERMXLACO:=FALSE
            ELSE
              BEGIN
%-------------------------AUMENTA O CONJUNTO W-----------------------%
                INTEGER K,CTOMIN,MAX;
                CTOMIN:=549755813887;
```

```
MAX:=-CTOMIN;
FOR K:=1 STEP 1 UNTIL L
DO
BEGIN
   INTEGER I,J,KK;
   INTEGER ARRAY SOMA[1:N];
   SOMA[V[K]]:=0;
   KK:=PRIXCOL[V[K]];
   DO
   BEGIN
      IF Y[A[1,KK]]+A[0,KK] <= 0
      THEN
        SOMA[V[K]]:=SOMA[V[K]]+
                    Y[A[1,KK]]+
                    A[0,KK];
      KK:=A[3,KK]
   END
   UNTIL KK = TERRA;
   IF SOMA[V[K]] > MAX
   THEN
     BEGIN
        MAX:=SOMA[V[K]];
        CTOMIN:=C[V[K]];
        JO:=V[K]
     END
   ELSE
     IF SOMA[V[K]] = MAX
     THEN
       IF C[V[K]] < CTOMIN
       THEN
         BEGIN
            CTOMIN:=C[V[K]];
            JO:=V[K]
         END
END;
BLSCOUXDISP:=TRUE;
```

```
                        NO:=DISPONIVEL;
                        DISPONIVEL:= - 1;
                        W[NO]:=JO;
                        X[JO]:=1-X[JO]
                      END
                  END
                ELSE
                  BEGIN
                    IF BUSCOLXOISP
                    THEN
                      DO
                      BEGIN
                        SUBLIXCOMPL;
                        IF ESGOTOUXW
                        THEN
                          PERMXLACO:=FALSE
                        ELSE
                          RESXSUB(A,AVANCA,B,C,DESTVDAS,
                                  DISPONIVEL,INDICE,NO,
                                  PRIXLIN,PRIXCOL,TERRA,W,X,
                                  ZA,ZERO,ZXMENOR)
                      END
                      UNTIL AVANCA >= 0 OR ¬ PERMXLACO
                    ELSE
                      PERMXLACO:=FALSE
                  END
              END
          END;
          END;
%-------------------------------------------------------------%
%------------INICIO DA ENUMERACAO IMPLICITA DAS PSEUDO-SOLUCOES
%------------PARCIAIS DE UM PROBLEMA DE PROGRAMACAO INTEIRA BI-
%------------VALENTE
%-------------------------------------------------------------%
   PROCEDURE ENUMXIMPL(A,B,C,CUSTOXNEG,INDICE,RESTRICAO,SOLXINI,
                       SOMA,TEMXSOLUCAO,TIPO,TRANSFORMOU,X,
```

```
                    XSOLXATUAL,Y,ZA,ZERO,ZXMENOR,ZXOTIMO);
          VALUE RESTRICAO,SOLXINI,TIPO;
          INTEGER CUSTOXNEG,RESTRICAO,SOLXINI,TIPO,ZA,ZXMENOR,
                  ZXOTIMO;
          INTEGER ARRAY A[1,1],B[1],C[1],INDICE[1],SOMA[1],
                        X[1],XSOLXATUAL[1],Y[1],ZERO[1];
          BOOLEAN TEMXSOLUCAO;
          BOOLEAN ARRAY TRANSFORMOU[1];
          BEGIN
             REAL AUX;
             ARRAY TESTE,SALVA[1:N];
             INTEGER AVANCA,DISPONIVEL,I,J,JO,KK,L,NO,SUBLINHADO;
             INTEGER ARRAY V,W[1:N];
             BOOLEAN CHAVE,BUSCOUXDISP,ESGOTOUXW,NEGATIVO,
                     PERMXLACO,VIAVEL,TROCA;
             BOOLEAN ARRAY ENT[1:N];
%-------------------------------------------------------------%
             PROCEDURE SUBLIXCOMPL;
                    BEGIN
                       INTEGER J;
                       J:=W[NC];
                       WHILE INDICE[J] = SUBLINHADO AND
                                       ¬ ESGOTOUXW
                       DO
                       BEGIN
                          INDICE[J]:=J;
                          X[J]:=0;
%-------------------------RETORNA O NO A FILHA DOS NOS
%-------------------------DISPONIVEIS
                          DISPONIVEL:=NC;
                          NO:= * - 1;
                          IF NO = C
                          THEN
                            ESGOTOUXW:=TRUE
                          ELSE
                            J:=W[NO]
```

```
                END;
                IF ¬ ESGOTOUX
                THEN
                  BEGIN
                     X[J]:=1-X[J];
                     IF ZERO[J] = 0
                     THEN
                       ZERO[J]:=J;
                     INDICE[J]:=SUBLINHADO
                  END
            END;
%-----------------------------------------------------------------%
        IF RESTRICAO = 1
        THEN
          FOR I:=1 STEP 1 UNTIL M
          DO
          FOR J:=1 STEP 1 UNTIL N
          DO
          A[I,J]:=-A[I,J];
        IF TIPO = 1
        THEN
          FOR J:=1 STEP 1 UNTIL N
          DO
          C[J]:=-C[J];
        CUSTOXNEG:=0;
        FOR J:=1 STEP 1 UNTIL N
        DO
        BEGIN
           INDICE[J]:=ZERO[J]:=J;
           TRANSFORMOU[J]:=FALSE;
           IF C[J] < 0
           THEN
             BEGIN
                CUSTOXNEG:= * + C[J];
                C[J]:=-C[J];
                TRANSFORMOU[J]:=TRUE;
```

```
                FOR I:=1 STEP 1 UNTIL M
                DO
                BEGIN
                   B[I]:=B[I]+A[I,J];
                   A[I,J]:=-A[I,J]
                END
             END
          END;
          DISPONIVEL:=1;
          SUBLINHADO:=ZXMENOR:=5497558138B7;
          TEMXSOLUCAO:=ESGOTOUXM:=FALSE;
          IF SOLXINI = 1
          THEN
            BEGIN
%--------------CRIACAO DE UMA SOLUCAO INICIAL--------------------------%
               FOR J:=1 STEP 1 UNTIL N
               DO
               BEGIN
                  FOR I:=1 STEP 1 UNTIL M
                  DO
                  SOMA[J]:=SOMA[J]+A[I,J];
                  TESTE[J]:=C[J]/SOMA[J];
                  KK:= * + 1;
                  SALVA[KK]:=TESTE[J]
               END;
               WRITE(SAIDA,<3(/)>);
               WRITE(SAIDA,</,X1,10F9.5>,
                             FOR J:=1 STEP 1 UNTIL N DO SALVA[J]);
               CHAVE:=TRUE;
               WHILE CHAVE
               DO
               BEGIN
                  KK:=0;
                  TROCA:=FALSE;
                  WHILE KK < M-1
                  DO
```

```
    BEGIN
       KK:= * + 1;
       IF SALVA[KK] > SALVA[KK+1]
       THEN
         BEGIN
            AUX:=SALVA[KK];
            SALVA[KK]:=SALVA[KK+1];
            SALVA[KK+1]:=AUX;
            TROCA:=TRUE
         END
    END;
    IF ¬ TROCA
    THEN
      CHAVE:=FALSE
END;
WRITE(SAIDA,<3(/)>);
WRITE(SAIDA,</,X1,10F9.5>,
              FOR J:=1 STEP 1 UNTIL N DO SALVA[J]);
FOR J:=1 STEP 1 UNTIL N
DO
ENT[J]:=FALSE;
FOR I:=1 STEP 1 UNTIL M
DO
Y[I]:=B[I];
KK:=0;
NEGATIVO:=TRUE;
WHILE NEGATIVO AND KK < N
DO
BEGIN
   KK:= * + 1;
   J:=0;
   DO
   J:= * + 1
   UNTIL SALVA[KK] = TESTE[J] AND ¬ ENT[J] OR J = N;
   IF SALVA[KK] = TESTE[J] AND ¬ ENT[J]
   THEN
```

```
                    BEGIN
                       BUSCOUXDISP:=TRUE;
                       NC:=DISPONIVEL;
                       DISPONIVEL:= * + 1;
                       W[NO]:=J;
                       X[J]:=1-X[J];
                       WRITE(SAIDA,<3(/)>);
                       WRITE(SAIDA,<X10,I4>,J);
                       ENT[J]:=TRUE;
                       FOR I:=1 STEP 1 UNTIL M
                       DO
                       Y[I]:=Y[I]+A[I,J]*X[J];
                       I:=0;
                       DO
                       I:= * + 1
                       UNTIL Y[I] < 0 OR I = M
                    END;
                    IF Y[I] >= 0
                    THEN
                      NEGATIVO:=FALSE
               END
%--------------FIM DA CRIACAO DE UMA SOLUCAO INICIAL-------------------%
             END;
          PERMXLACO:=TRUE;
          WHILE PERMXLACO
          DO
          BEGIN
             INTEGER CJMIN;
%------------CALCULO DO VETOR Y-----------------------------------------%
             INTEGER I;
             FOR I:=1 STEP 1 UNTIL M
             DO
             BEGIN
                INTEGER J,SOMA;
                SOMA:=0;
                FOR J:=1 STEP 1 UNTIL N
```

```
                DO
                SOMA:=SOMA+A[I,J]*X[J];
                Y[I]:=B[I]+SOMA
            END;
            I:=0;
            DO
            I:= * + 1
            UNTIL Y[I] < 0  OR I = N;
            IF Y[I] >= 0
            THEN
              BEGIN
%-----------------CALCULO DA FUNCAO OBJETIVO-----------------------------%
               INTEGER J;
               ZA:=0;
               FOR J:=1 STEP 1 UNTIL N
               DO
               ZA:=ZA+C[J]*X[J];
               IF ZA < ZXMENOR
               THEN
                 BEGIN
                    TEMXSOLUCAO:=TRUE;
                    ZXMENOR:=ZA;
                    WRITE(SAIDA,</,X10,I10>,ZXMENOR);
                    WRITE(SAIDA,</,X1,25I5,/>,
                                 FOR J:=1 STEP 1 UNTIL N DO X[J]);
                    FOR J:=1 STEP 1 UNTIL N
                    DO
                    XSOLXATUAL[J]:=X[J]
                 END;
               IF BUSCOUXDISP
               THEN
                    DO
                    BEGIN
                       SUBLIXCOMPL;
                       IF ESGOTOUXW
                       THEN
```

```
                    PERMXLACO:=FALSE
                  ELSE
                    RESTRICXSCA,AVANCA,B,C,DISPONIVEL,
                              INDICE,NO,k,X,ZA,ZERO,
                              ZXMENOR);
                END
                UNTIL AVANCA >= 0 OR ¬ PERMXLACO
              ELSE
                PERMXLACO:=FALSE
            END
          ELSE
            BEGIN
%-----------------CONSTRUCAO DO CONJUNTO V---------------------%
              INTEGER I,J,K;
              BOOLEAN ARRAY ENTROU[1:N];
              FOR J:=1 STEP 1 UNTIL N
              DO
              ENTROU[J]:=FALSE;
              K:=L:=0;
              ZA:=0;
              CJMIN:=5497558138B7;
              FOR J:=1 STEP 1 UNTIL N
              DO
              ZA:=ZA+C[J]*X[J];
              FOR I:=1 STEP 1 UNTIL M
              DO
              IF Y[I] < 0
              THEN
                FOR J:=1 STEP 1 UNTIL N
                DO
                IF X[J] = 0 AND INDICE[J] = J
                            AND ¬ ENTROU[J]
                            AND ZERO[J] = J
                THEN
                  IF ZA+C[J] < ZXMENOR AND A[I,J] > 0
                  THEN
```

```
                    BEGIN
                       K:= * + 1;
                       V[K]:=J;
                       ENTROU[J]:=TRUE;
                       L:=K;
                       IF C[J] < CJMIN
                       THEN
                         CJMIN:=C[J]
                    END;
               IF L ¬= 0
               THEN
                 BEGIN
%----------------------TESTA VIABILIDADE-----------------------------------%
                    INTEGER I,J;
                    INTEGER ARRAY SOMA[1:M];
                    VIAVEL:=TRUE;
                    I:=0;
                    DO
                    BEGIN
                       I:= * + 1;
                       SOMA[I]:=0;
                       IF Y[I] < 0
                       THEN
                         BEGIN
                            INTEGER K;
                            FOR K:=1 STEP 1 UNTIL L
                            DO
                            BEGIN
                               J:=V[K];
                               IF A[I,J] > 0
                               THEN
                                 SOMA[I]:=SOMA[I]+A[I,J]
                            END;
                            SOMA[I]:=Y[I]+SOMA[I]
                         END
                    END
```

```
                UNTIL SOMA[I] < 0 OR I = M;
                IF SOMA[I] < 0
                THEN
                  VIAVEL:=FALSE;
                IF ¬ VIAVEL
                THEN
                  IF BUSCOUXDISP
                  THEN
                    DO
                    BEGIN
                       SUBLIXCOMPL;
                       IF ESGOTOUXk
                       THEN
                         PERMXLACO:=FALSE
                       ELSE
                         RESTRICXS(A,AVANCA,B,C,
                                   DISPONIVEL,INDICE,
                                   NC,W,X,ZA,ZERO,
                                   ZXMENOR)
                    END
                    UNTIL AVANCA >= 0 OR ¬ PERMXLACO
                  ELSE
                    PERMXLACO:=FALSE
                ELSE
                  BEGIN
%-------------------------AUMENTA O CONJUNTO k----------------%
                     INTEGER AT,SUM,MAX;
                     BOOLEAN ARRAY ESCOLHIDO[1:N];
                     AT:=0;
                     SUM:=ZA;
                     FOR K:=1 STEP 1 UNTIL L
                     DO
                     ESCOLHIDO[V[K]]:=FALSE;
                     DO
                     BEGIN
                        INTEGER K,CTOMIN;
```

```
CTOMIN:=549755813887;
MAX:=-CTOMIN;
AT:= * + 1;
FOR K:=1 STEP 1 UNTIL L
DO
BEGIN
   INTEGER I,J;
   INTEGER ARRAY SOMA[1:N];
   J:=V[K];
   IF ¬ ESCOLHIDO[J]
   THEN
     BEGIN
        SOMA[J]:=0;
        FOR I:=1 STEP 1 UNTIL M
        DO
        IF Y[I]+A[I,J] <= 0
        THEN
          SOMA[J]:=SOMA[J]+Y[I]+A[I,J];
        IF SOMA[J] > MAX
        THEN
          BEGIN
             MAX:=SOMA[J];
             CTOMIN:=C[J];
             JO:=J
          END
        ELSE
          IF SOMA[J] = MAX
          THEN
            IF C[J] < CTOMIN
            THEN
              BEGIN
                 CTOMIN:=C[J];
                 JO:=J
              END
     END
END;
```

```
                ESCOLHIDO[JO]:=TRUE;
                SUM:= * + C[JO];
                BUSCOUXDISP:=TRUE;
                NO:=DISPONIVEL;
                DISPONIVEL:= * + 1;
                W[NO]:=JO;
                X[JO]:=1-X[JO];
                FOR I:=1 STEP 1 UNTIL M
                DO
                Y[I]:=Y[I]+A[I,JO]
            END
            UNTIL SUM+CJMIN >= ZXMENOR OR AT = L
                                    OR MAX = 0;
        END
    END
ELSE
    BEGIN
        IF BUSCOUXDISP
        THEN
            DO
            BEGIN
                SUBLIXCOMPL;
                IF ESGOTOUXW
                THEN
                    PERMXLACO:=FALSE
                ELSE
                    RESTRICXS(A,AVANCA,B,C,
                              DISPONIVEL,INDICE,
                              NO,W,X,ZA,ZERO,
                              ZXMENOR)
            END
            UNTIL AVANCA >= 0 OR ¬ PERMXLACO
        ELSE
            PERMXLACO:=FALSE
    END
END
```

```
        END;
    END;
%----------------------FIM DA ENUMXIMPL-----------------------%
%-------------------------------------------------------------%
    PROCEDURE REDUCOES(A,C,PARTICI,PRIXLIN,PRIXCOL,RECOBRE,
                       TERRA,X);
        VALUE PARTICI,RECOBRE,TERRA;
        INTEGER PARTICI,RECOBRE,TERRA;
        INTEGER ARRAY A[0,1],C[1],PRIXLIN[1],PRIXCOL[1],X[1];
        BEGIN
            INTEGER I,IK,J,JC,JK,K,KK,K1,K2,KK1,KK2,L;
            BOOLEAN CNDC1,CNDC2,DESAT1,DESAT2,DESAT3,DESAT4,
                    DESAT5,DESAT6,DESAT7,DESAT8,DESAT9,
                    DESAT10,DESAT11,DESAT12,DESAT13;
%-------------------------------------------------------------%
            PROCEDURE DESATIVAXLINHA(I);
                VALUE I;
                INTEGER I;
                BEGIN
                    INTEGER J,K,P;
                    PRIXLIN[I]:=TERRA;
                    FOR J:=1 STEP 1 UNTIL N
                    DO
                    IF PRIXCOL[J] ¬= TERRA
                    THEN
                      BEGIN
                        P:=0;
                        K:=PRIXCOL[J];
                        WHILE A[1,K] ¬= I AND A[3,K] ¬= TERRA
                        DO
                        BEGIN
                            P:=K;
                            K:=A[3,K]
                        END;
                        IF P ¬= 0
                        THEN
```

```
                    IF A[3,K] ¬= TERRA
                    THEN
                      A[3,P]:=A[3,K]
                    ELSE
                      IF A[1,K] = I
                      THEN
                        A[3,P]:=TERRA
                      ELSE
                  ELSE
                    IF A[3,K] ¬= TERRA
                    THEN
                      PRIXCOL[J]:=A[3,K]
                    ELSE
                      PRIXCOL[J]:=TERRA
              END
          END;
%-------------------------------------------------------------------%
      PROCEDURE DESATIVAXCOLUNA(J);
          VALUE J;
          INTEGER J;
          BEGIN
             INTEGER I,K,P;
             PRIXCOL[J]:=TERRA;
             FOR I:=1 STEP 1 UNTIL MK
             DO
             IF PRIXLIN[I] ¬= TERRA
             THEN
               BEGIN
                  P:=0;
                  K:=PRIXLIN[I];
                  WHILE A[2,K] ¬= J AND A[4,K] ¬= TERRA
                  DO
                  BEGIN
                     P:=K;
                     K:=A[4,K]
                  END;
```

```
                        IF P ¬= 0
                        THEN
                          IF A[4,K] ¬= TERRA
                          THEN
                            A[4,P]:=A[4,K]
                          ELSE
                            IF A[2,K] = J
                            THEN
                              A[4,P]:=TERRA
                            ELSE
                        ELSE
                          IF A[4,K] ¬= TERRA
                          THEN
                            PRIXLIN[I]:=A[4,K]
                          ELSE
                            PRIXLIN[I]:=TERRA
                    END
                END;
%------------------INICIO DAS REDUCOES-------------------------%
        DO
        BEGIN
          DESAT1:=DESAT2:=DESAT3:=DESAT4:=DESAT5:=DESAT6:=
          DESAT7:=DESAT8:=DESAT9:=DESAT10:=DESAT11:=DESAT12:=
          DESAT13:=FALSE;
          FOR I:=1 STEP 1 UNTIL MK
          DO
          IF PRIXLIN[I] ¬= TERRA
          THEN
            BEGIN
              J:=PRIXLIN[I];
              IF A[4,J] = TERRA
              THEN
                BEGIN
                  X[A[2,J]]:=1;
                  K:=PRIXCCL[A[2,J]];
                  WHILE K ¬= TERRA
```

```
                    DO
                    BEGIN
                       IF PARTICI = 3
                       THEN
                         BEGIN
                            JK:=A[3,K];
                            WHILE JK ¬= TERRA
                            DO
                            BEGIN
                               KK:=PRIXLIN[A[1,JK]];
                               WHILE KK ¬= TERRA
                               DO
                               BEGIN
                                  JJ:=A[2,KK];
                                  DESATIVAXCOLUNA(JJ);
                                  DESAT1:=TRUE;
                                  KK:=A[4,KK]
                               END;
                               JK:=A[3,JK]
                            END
                         END;
                       KK:=A[1,K];
                       DESATIVAXLINHA(KK);
                       DESAT2:=TRUE;
                       K:=A[3,K]
                    END;
                    KK:=A[2,J];
                    DESATIVAXCOLUNA(KK);
                    DESAT3:=TRUE
                 END
              END;
%-------------------------------------------------------------------%
              FOR I:=1 STEP 1 UNTIL MK-1
              DO
              BEGIN
                 L:=I;
```

```
FOR I:=I+1 STEP 1 UNTIL MK
DO
BEGIN
   K1:=PRIXLIN[L];
   K2:=PRIXLIN[I];
   IF K1 ¬= TERRA AND K2 ¬= TERRA
   THEN
     BEGIN
        CNDC1:=CNDC2:=TRUE;
        WHILE CNDC1 AND CNDC2
        DO
        IF A[2,K1] = A[2,K2]
        THEN
          BEGIN
             KK1:=K1;
             KK2:=K2;
             K1:=A[4,K1];
             K2:=A[4,K2];
             IF K1 = TERRA OR K2 = TERRA
             THEN
               CNDC1:=FALSE
          END
        ELSE
          IF A[2,K1] < A[2,K2]
          THEN
            BEGIN
               KK1:=K1;
               K1:=A[4,K1];
               IF K1 = TERRA
               THEN
                 CNDC1:=FALSE
            END
          ELSE
            IF A[2,K1] > A[2,K2]
            THEN
              CNDC2:=FALSE;
```

```
IF CNDC2 AND K2 = TERRA
THEN
  BEGIN
      IF PARTICI = 3
      THEN
        BEGIN
           KK:=PRIXLIN[A[1,KK1]];
           JJ:=PRIXLIN[A[1,KK2]];
           WHILE KK <> TERRA AND JJ <> TERRA
           DO
           BEGIN
              IK:=KK;
              IF A[2,KK] < A[2,JJ]
              THEN
                BEGIN
                   JK:=A[2,KK];
                   DESATIVAXCOLUNA(JK);
                   DESAT4:=TRUE;
                   KK:=A[4,KK]
                END;
              IF A[2,IK] = A[2,JJ]
              THEN
                BEGIN
                   KK:=A[4,KK];
                   JJ:=A[4,JJ]
                END
              ELSE
                IF A[2,IK] > A[2,JJ]
                THEN
                  JJ:=A[4,JJ]
           END;
           WHILE KK <> TERRA AND JJ = TERRA
           DO
           BEGIN
              JK:=A[2,KK];
              DESATIVAXCOLUNA(JK);
```

```
                                DESAT5:=TRUE;
                                KK:=A[4,KK]
                            END
                        END;
                    KK:=A[1,KK1];
                    DESATIVAXLINHA(KK);
                    DESAT6:=TRUE
                END
            END
        END
    END;
%-----------------------------------------------------------------------%
    FOR I:=MK STEP -1 UNTIL 2
    DO
    BEGIN
        L:=I;
        FOR I:=I-1 STEP -1 UNTIL 2
        DO
        BEGIN
            K1:=PRIXLIN[I];
            K2:=PRIXLIN[L];
            IF K1 ¬= TERRA AND K2 ¬= TERRA
            THEN
              BEGIN
                CNDC1:=CNDC2:=TRUE;
                WHILE CNDC1 AND CNDC2
                DO
                IF A[2,K1] = A[2,K2]
                THEN
                  BEGIN
                    KK1:=K1;
                    KK2:=K2;
                    K1:=A[4,K1];
                    K2:=A[4,K2];
                    IF K1 = TERRA OR K2 = TERRA
                    THEN
```

```
          CNDC1:=FALSE
    END
  ELSE
    IF A[2,K1] < A[2,K2]
    THEN
      BEGIN
         KK1:=K1;
         K1:=A[4,K1];
         IF K1 = TERRA
         THEN
           CNDC1:=FALSE
      END
    ELSE
      IF A[2,K1] > A[2,K2]
      THEN
        CNDC2:=FALSE;
  IF CNDC2 AND K2 = TERRA
  THEN
    BEGIN
       IF PARTICI = 3
       THEN
         BEGIN
            KK:=PRIXLIN[A[1,KK1]];
            JJ:=PRIXCOL[A[1,KK2]];
            WHILE KK <> TERRA AND JJ <> TERRA
            DO
            BEGIN
               IK:=KK;
               IF A[2,KK] < A[2,JJ]
               THEN
                 BEGIN
                    JK:=A[2,KK];
                    DESATIVAXCOLUNA(JK);
                    DESAT7:=TRUE;
                    KK:=A[4,KK]
                 END;
```

```
                        IF A[2,IK] = A[2,JJ]
                        THEN
                          BEGIN
                             KK:=A[4,KK];
                             JJ:=A[4,JJ]
                          END
                        ELSE
                          IF A[2,IK] > A[2,JJ]
                          THEN
                            JJ:=A[4,JJ]
                      END;
                    WHILE KK ¬= TERRA AND JJ = TERRA
                     DO
                     BEGIN
                        JK:=A[2,KK];
                        DESATIVAXCOLUNA(JK);
                        DESAT8:=TRUE;
                        KK:=A[4,KK]
                     END
                  END;
               KK:=A[1,KK1];
               DESATIVAXLINHA(KK);
               DESAT9:=TRUE
            END
         END
      END
   END;
%-------------------------------------------------------------------%
      FOR J:=1 STEP 1 UNTIL N-1
      DO
      BEGIN
         L:=J;
         FOR K:=J+1 STEP 1 UNTIL N
         DO
         BEGIN
            CNDC1:=TRUE;
```

```
                CNDC2:=FALSE;
                K1:=PRIXCOL[L];
                K2:=PRIXCOL[K];
                WHILE K1 ¬= TERRA AND K2 ¬= TERRA AND CNDC1
                DO
                IF A[1,K1] = A[1,K2]
                THEN
                  BEGIN
                     CNDC2:=TRUE;
                     K1:=A[3,K1];
                     K2:=A[3,K2]
                  END
                ELSE
                  CNDC1:=FALSE;
                IF K1 = TERRA AND K2 = TERRA AND CNDC1
                                             AND CNDC2
                THEN
                  IF C[L] <= C[K]
                  THEN
                    BEGIN
                       DESATIVAXCOLUNA(K);
                       DESAT10:=TRUE
                    END
                  ELSE
                    BEGIN
                       DESATIVAXCOLUNA(L);
                       DESAT11:=TRUE
                    END
              END
            END;
%-------------------------------------------------------------------%
            IF RECOBRE = 2
            THEN
              FOR J:=1 STEP 1 UNTIL N-1
              DO
              BEGIN
```

```
L:=J;
FOR K:=J+1 STEP 1 UNTIL N
DO
BEGIN
   CNDC1:=TRUE;
   CNDC2:=FALSE;
   IF C[L] <= C[K]
   THEN
     BEGIN
        K1:=PRIXCOL[L];
        K2:=PRIXCOL[K];
        WHILE K1 ¬= TERRA AND K2 ¬= TERRA
                              AND CNDC1
        DO
        BEGIN
           IF A[1,K1] = A[1,K2]
           THEN
             BEGIN
                K1:=A[3,K1];
                K2:=A[3,K2]
             END
           ELSE
             IF A[1,K1] < A[1,K2]
             THEN
               BEGIN
                  CNDC2:=TRUE;
                  K1:=A[3,K1]
               END
             ELSE
               CNDC1:=FALSE
        END;
        IF K2 = TERRA AND CNDC1 AND CNDC2
        THEN
          BEGIN
             DESATIVAXCOLUNA(K);
             DESAT12:=TRUE
```

```
        END
    END;
  CNDC1:=TRUE;
  CNDC2:=FALSE;
  IF C[K] <= C[L]
  THEN
    BEGIN
       K1:=PRIXCOL[K];
       K2:=PRIXCOL[L];
       WHILE K1 <> TERRA AND K2 <> TERRA
                          AND CNDC1
       DO
       BEGIN
          IF A[1,K1] = A[1,K2]
          THEN
            BEGIN
               K1:=A[3,K1];
               K2:=A[3,K2]
            END
          ELSE
            IF A[1,K1] < A[1,K2]
            THEN
              BEGIN
                 CNDC2:=TRUE;
                 K1:=A[3,K1]
              END
            ELSE
              CNDC1:=FALSE
       END;
       IF K2 = TERRA AND CNDC1 AND CNDC2
       THEN
         BEGIN
            DESATIVAXCOLUNA(L);
            DESAT13:=TRUE
         END
    END
```

```
                    END
                END
            END
            UNTIL ¬ DESAT1  AND ¬ DESAT2  AND ¬ DESAT3  AND
                  ¬ DESAT4  AND ¬ DESAT5  AND ¬ DESAT6  AND
                  ¬ DESAT7  AND ¬ DESAT8  AND ¬ DESAT9  AND
                  ¬ DESAT10 AND ¬ DESAT11 AND ¬ DESAT12 AND
                  ¬ DESAT13
        END;
%-------------------------------------------------------------%
    PROCEDURE AJUSXPONT(A,LINXSEG,PRIXLIN,PRIXCOL,TERRA,X);
        INTEGER TERRA;
        INTEGER ARRAY A[0,1],LINXSEG[0],PRIXLIN[1],PRIXCOL[1],
                      X[1];
        BEGIN
            INTEGER I,J;
            TERRA:=5497558138887;
            FOR I:=1 STEP 1 UNTIL ENTIER(M*N/2)-1
            DO
            A[4,I]:=I+1;
            A[4,ENTIER(M*N/2)]:=TERRA;
            FOR I:=1 STEP 1 UNTIL M
            DO
            PRIXLIN[I]:=TERRA;
            FOR J:=1 STEP 1 UNTIL N
            DO
            PRIXCOL[J]:=TERRA;
            FOR I:=0 STEP 1 UNTIL M-1
            DO
            LINXSEG[I]:=I+1;
            LINXSEG[M]:=TERRA;
            FOR J:=1 STEP 1 UNTIL N
            DO
            X[J]:=0;
        END;
%-------------------------------------------------------------%
```

```
PROCEDURE ATIVXPONT(A,LINHA,COLUNA,PONTEIRO,PRIXLIN,PRIXCOL,
                                    TERRA,X);
        VALUE COLUNA,LINHA,PONTEIRO,TERRA;
        INTEGER COLUNA,LINHA,PONTEIRO,TERRA;
        INTEGER ARRAY A[0,1],PRIXLIN[1],PRIXCOL[1],X[1];
        BEGIN
           INTEGER H,I,J,K,XK;
%-------------------------------------------------------------%
           PROCEDURE LIGAXPONTXLINHA;
                 BEGIN
                    A[4,PONTEIRO]:=K;
                    IF H = TERRA
                    THEN
                      PRIXLIN[LINHA]:=PONTEIRO
                    ELSE
                      A[4,H]:=PONTEIRO;
                    XK:=1
                 END;
%-------------------------------------------------------------%
           PROCEDURE LIGAXPONTXCOLUNA;
                 BEGIN
                    A[3,PONTEIRO]:=K;
                    IF H = TERRA
                    THEN
                      PRIXCOL[COLUNA]:=PONTEIRO
                    ELSE
                      A[3,H]:=PONTEIRO;
                    XK:=0
                 END;
%-------------------------------------------------------------%
%------------ATIVA PONTEIROS NAS LINHAS-----------------------%
              H:=TERRA;
              K:=PRIXLIN[LINHA];
              IF K =: H
              THEN
                FOR J:=1 STEP 1 UNTIL N
```

```
                DO
                IF XK = 0
                THEN
                  IF A[2,K] < COLUNA
                  THEN
                    BEGIN
                       H:=K;
                       K:=A[4,K];
                       IF K = TERRA
                       THEN
                         LIGAXPONTXLINHA
                    END
                  ELSE
                    LIGAXPONTXLINHA
                ELSE
                  BEGIN
                     A[4,PONTEIRO]:=H;
                     PRIXLIN[LINHA]:=PONTEIRO;
                     XK:=1
                  END;
%---------------ATIVA PONTEIROS NAS COLUNAS-----------------------------%
                H:=TERRA;
                K:=PRIXCOL[COLUNA];
                IF K ¬= H
                THEN
                  FOR I:=1 STEP 1 UNTIL M
                  DO
                  IF XK = 1
                  THEN
                    IF A[1,K] < LINHA
                    THEN
                      BEGIN
                         H:=K;
                         K:=A[3,K];
                         IF K = TERRA
```

```
                    THEN
                      LIGAXPONTXCOLUNA
                  END
                ELSE
                  LIGAXPONTXCCLUNA
              ELSE
            ELSE
              BEGIN
                 A[3,PONTEIRO]:=H;
                 PRIXCOL[COLUNA]:=PONTEIRO;
                 XK:=0
              END
        END;
%-------------------------------------------------------------%
   PROCEDURE REAJXPONT(LINXSEG,PRIXLIN,TERRA);
        VALUE TERRA;
        INTEGER TERRA;
        INTEGER ARRAY LINXSEG[0],PRIXLIN[1];
        BEGIN
           INTEGER I,L;
           I:=0;
           DO
           BEGIN
              DO
              I:= * + 1
              UNTIL PRIXLIN[I] = TERRA OR I = M;
              IF PRIXLIN[I] = TERRA AND I ¬= M
              THEN
                BEGIN
                   L:=I;
                   DO
                   L:= * + 1
                   UNTIL PRIXLIN[L] ¬= TERRA OR
                                      L = M;
                   IF PRIXLIN[L] ¬= TERRA
                   THEN
```

```
                    BEGIN
                       LINXSEG[I-1]:=LINXSEG[L-1];
                       I:=L
                    END
                 ELSE
                    IF L = M
                    THEN
                      BEGIN
                         LINXSEG[I-1]:=TERRA;
                         I:=L
                      END
              END
           ELSE
              IF PRIXLIN[I] = TERRA AND I = M
              THEN
                LINXSEG[I-1]:=TERRA
        END
        UNTIL I = M
     END;
%-------------------------------------------------------------------%
  PROCEDURE ESCREXDADOS(A,B,C,LINXSEG,PRIXLIN,PRIXCOL,TERRA);
        VALUE TERRA;
        INTEGER TERRA;
        INTEGER ARRAY A[0,1],B[1],C[1],LINXSEG[0],PRIXLIN[1],
                      PRIXCOL[1];
        BEGIN
           INTEGER I,J,K;
           INTEGER ARRAY MATRIZ[1:MK,1:N];
           FOR I:=1 STEP 1 UNTIL MK
           DO
           FOR J:=1 STEP 1 UNTIL N
           DO
           MATRIZ[I,J]:=0;
           FOR I:=1 STEP 1 UNTIL MK
           DO
           BEGIN
```

```
        K:=PRIXLIN[I];
        WHILE K ¬= TERRA
        DO
        BEGIN
           INTEGER LINHA,COLUNA;
           LINHA:=A[1,K];
           COLUNA:=A[2,K];
           K:=A[4,K];
           MATRIZ[LINHA,COLUNA]:=1
        END
     END;
     WRITE(SAIDA,<//,X10,"CUSTOS",//>);
     WRITE(SAIDA,<X1,25I5,/>,FOR J:=1 STEP 1 UNTIL N DO C[J]);
     WRITE(SAIDA,<2(/)>);
     WRITE(SAIDA,<//,X10,"MATRIZ DO MODELO",//>);
     I:=LINXSEG[0];
     DO
     BEGIN
        IF PRIXLIN[I] ¬= TERRA
        THEN
          BEGIN
             WRITE(SAIDA,<2(/)>);
             WRITE(SAIDA,<X1,25I5,/,>,FOR J:=1 STEP 1
                                      UNTIL N DO
                                      IF PRIXCOL[J] ¬=
                                      TERRA THEN
                                      MATRIZ[I,J]);
          END;
        I:=LINXSEG[I]
     END
     UNTIL I = TERRA;
     WRITE(SAIDA,<2(/)>);
     WRITE(SAIDA,<//,X10,"VETOR CONSTANTE",//>);
     WRITE(SAIDA,<X1,25I5,/>,FOR I:=1 STEP 1 UNTIL MK
                             DO B[I])
END;
```

```
%-------------------------------------------------------------%
   PROCEDURE ESCREXRESUL(C,TEMXSOLUCAO,X,XSOLXATUAL,ZA,ZXOTIMO);
          INTEGER ZA,ZXOTIMO;
          INTEGER ARRAY C[1],X[1],XSOLXATUAL[1];
          BOOLEAN TEMXSOLUCAO;
          BEGIN
             INTEGER I,J;
             ZA:=0;
             FOR J:=1 STEP 1 UNTIL N
             DO
             BEGIN
                X[J]:=XSOLXATUAL[J];
                ZA:=ZA+C[J]*X[J]
             END;
             ZXOTIMO:=ZA;
             IF TEMXSOLUCAO
             THEN
               BEGIN
                  WRITE(SAIDA,<//,X10,
 "INDICES DAS VARIAVEIS IGUAIS A 1 NA SOLUCAO",//>);
                  WRITE(SAIDA,<//,X1,25I5,/>,FOR J:=1 STEP 1
                                UNTIL N DO IF X[J] = 1 THEN J);
                  WRITE(SAIDA,<//,X10,"VETOR SOLUCAO",//>);
                  WRITE(SAIDA,<//,X1,25I5,/>,FOR J:=1 STEP 1
                                UNTIL N DO X[J]);
                  WRITE(SAIDA,<//,X10,"VALOR OTIMO DA F. O.",
                                I10,//,130("="),//>,ZXOTIMO)
               END
             ELSE
               WRITE(SAIDA,<//,X10,"O PROBLEMA NAO TEM SOLUCAO",
                             //,130("="),//>)
          END;
%-------------------------------------------------------------%
   PROCEDURE RETRXMEMO(A,LINXSEG,PRIXLIN,REDUC,TERRA);
          VALUE REDUC,TERRA;
          INTEGER REDUC,TERRA;
```

```
        INTEGER ARRAY A[0:1],LINXSEG[0],PRIXLIN[1];
        BEGIN
           INTEGER I,K;
           WRITE(SAIDA,<2(/)>);
           WRITE(SAIDA,<//,
           X60,"DISPOSICAO DOS DADOS NA MEMORIA">);
           WRITE(SAIDA,<3(/)>);
           FOR K:=1 STEP 1 WHILE A[0,K] ¬= 0
           DO
           WRITE(SAIDA,</,X20,6I15>,K,A[0,K],A[1,K],A[2,K],
                                        A[3,K],A[4,K]);
           WRITE(SAIDA,<3(/)>);
           IF REDUC = 0
           THEN
             BEGIN
                K:=0;
                WHILE LINXSEG[K] ¬= TERRA
                DO
                BEGIN
                   K:=LINXSEG[K];
                   I:=PRIXLIN[K];
                   WHILE I ¬= TERRA
                   DO
                   BEGIN
                      WRITE(SAIDA,</,X20,6I15,>,I,A[0,I],A[1,I],
                                   A[2,I],A[3,I],A[4,I]);
                      I:=A[4,I]
                   END
                END
             END;
        END;
%-------------------------------------------------------------%
   PROCEDURE ENTRADAXSAIDA;
        BEGIN
           INTEGER ENUMERA,EMPAREL,PARTICI,RECOBRE,RECXGRA,
                   REDUC;
```

```
            READ(ENTRADA,<5I1,X9,I1>,ENUMERA,RECOBRE,PARTICI,
                                        RECXGRA,EMPAREL,REDUC);
            IF ENUMERA = 1
            THEN
              BEGIN
                 INTEGER CUSTOXNEG,I,J,RESTRICAO,SOLXINI,TIPO,
                         ZA,ZXMENOR,ZXOTIMO;
                 INTEGER ARRAY A[1:M,1:N],B[1:M],C[1:N],
                               INDICE[1:N],SOMA[1:N],X[1:N],
                               XSOLXATUAL[1:N],Y[1:M],ZERO[1:N];
                 BOOLEAN TEMXSOLUCAO;
                 BOOLEAN ARRAY TRANSFORMCU[1:N];
                 READ(ENTRADA,<3I1>,RESTRICAO,SOLXINI,TIPO);
%----------------LEITURA DOS CUSTOS
                 READ(ENTRADA,<16I5>,FOR J:=1 STEP 1 UNTIL N
                                        DO C[J]);
%----------------LEITURA DA MATRIZ DO MODELO
                 FOR I:=1 STEP 1 UNTIL M
                 DO
                 READ(ENTRADA,<16I5>,FOR J:=1 STEP 1 UNTIL N DO
                                        A[I,J]);
%----------------LEITURA DO VETOR B
                 READ(ENTRADA,<16I5>,FOR I:=1 STEP 1 UNTIL M DO
                                        B[I]);
%----------------ESCREVE OS CUSTOS, A MATRIZ DO MODELO E
%----------------O VETOR B
                 WRITE(SAIDA,<2(/)>);
                 WRITE(SAIDA,<//,X10,"CUSTOS",//>);
                 WRITE(SAIDA,<X1,18I7,/>,FOR J:=1 STEP 1 UNTIL
                                           N DO C[J]);
                 WRITE(SAIDA,<2(/)>);
                 WRITE(SAIDA,<//,X10,"MATRIZ DO MODELO",//>);
                 FOR I:=1 STEP 1 UNTIL M
                 DO
                 BEGIN
                    WRITE(SAIDA,<2(/)>);
```

```
                WRITE(SAIDA,</,X1,18I7,/>,FOR J:=1 STEP 1
                                          UNTIL N DO A[I,J])
            END;
            WRITE(SAIDA,<2(/)>);
            WRITE(SAIDA,<//,X10,"ELEMENTOS DO VETOR B",//>);
            WRITE(SAIDA,<2(/)>);
            WRITE(SAIDA,</,X1,18I7,/>,FOR I:=1 STEP 1
                                      UNTIL M DO B[I]);
            WRITE(SAIDA,<2(/)>);
            ENUMXIMPL(A,B,C,CUSTOXNEG,INDICE,RESTRICAO,
                      SOLXINI,SOMA,TEMXSOLUCAO,TIPO,
                      TRANSFORMOU,X,XSOLXATUAL,Y,ZA,ZERO,
                      ZXMENOR,ZXOTIMO);
%-----------------ESCREVE OS RESULTADOS
            ZA:=0;
            FOR J:=1 STEP 1 UNTIL N
            DO
            BEGIN
               X[J]:=XSOLXATUAL[J];
               ZA:=ZA+C[J]*X[J];
               IF TRANSFORMOU[J]
               THEN
                 X[J]:=1-X[J]
            END;
            ZXOTIMO:=ZA+CUSTOXNEG;
            IF TIPO = 1
            THEN
              ZXOTIMO:=-ZXOTIMO;
            IF TEMXSOLUCAO
            THEN
              BEGIN
                 WRITE(SAIDA,<//,X10,
"SOLUCAO DO PROB. PELO METODO DE ENUMERACAO IMPLICITA",
                                                    //>);
                 WRITE(SAIDA,<//,X15,
"INDICE DAS VARIAVEIS IGUAIS A 1 NA SOLUCAO",
```

```
                                                        //>);
                        WRITE(SAIDA,<//,X1,25I5,/>,FOR J:=1 STEP
                              1 UNTIL N DO IF X[J] = 1 THEN J);
                        WRITE(SAIDA,<//,X15,
   "VETOR SOLUCAO",
                                                         //>);
                        WRITE(SAIDA,<//,X1,25I5,/>,FOR J:=1 STEP
                                    1 UNTIL N DO X[J]);
                        WRITE(SAIDA,<//,X15,
   "VALOR OTIMO DA FUNCAO OBJETIVO",I10,//,130("="),//>,ZXOTIMO)
                     END
                   ELSE
                     WRITE(SAIDA,<//,X10,
  "O PROBLEMA PROPOSTO PARA SER RESOLVIDO PELO METODO DE",//,X10
   ,"ENUMERACAO IMPLICITA, NAO TEM SOLUCAO",//,130("="),//>)
                END;
              IF RECOBRE = 2 OR RECXGRA = 4 OR EMPAREL = 5
              THEN
                BEGIN
                   INTEGER CSTXNEG,DISPO,I,J,LINHA,COLUNA,
                           PONTEIRO,TERRA,ZA,ZXMENOR,ZXOTIMO;
                   INTEGER ARRAY A[0:4,1:ENTIER(M*N/2)],B[1:M],
                                C[1:N],INDICE[1:N],LINXSEG[0:M],
                                PRIXLIN[1:M],PRIXCOL[1:N],
                                SOMA[1:N],X[1:N],XSOLXATUAL[1:N],
                                Y[1:M],ZERO[1:N];
                   BOOLEAN TEMXSOLUCAO;
                   MK:=M;
                   AJUSXFONT(A,LINXSEG,PRIXLIN,PRIXCOL,TERRA,X);
%-----------------LEITURA DOS CUSTOS
                   READ(ENTRADA,<16I5>,FOR J:=1 STEP 1 UNTIL N
                                       DO C[J]);
                   DISPO:=1;
%-----------------LEITURA DA MATRIZ A
                   READ(ENTRADA,<2I4>,LINHA,COLUNA);
                   WHILE LINHA ¬= 0 AND DISPO ¬= TERRA
```

```
                    DO
                    BEGIN
                       PONTEIRO:=DISPO;
                       A[0,PONTEIRO]:=1;
                       A[1,PONTEIRO]:=LINHA;
                       A[2,PONTEIRO]:=COLUNA;
                       DISPO:=A[4,DISPO];
                       ATIVXPONT(A,LINHA,COLUNA,PONTEIRO,PRIXLIN,
                                 PRIXCOL,TERRA,X);
                       READ(ENTRADA,<2I4>,LINHA,COLUNA)
                    END;
                    IF DISPO = TERRA
                    THEN
                      WRITE(SAIDA,<// X10,
 "FORAM ESGOTADAS AS POSICOES DA MATRIZ, PROVAVELMENTE ELA",//,
 X10,"NAO E ESPARSA. TENTE UTILIZAR A PROCEDURE ENUMXIMPL.",//>)
                    ELSE
                      BEGIN
%---------------------LEITURA DO VETOR CONSTANTE
                        READ(ENTRADA,<16I5>,FOR I:=1 STEP 1 UNTIL
                                            M DO B[I]);
                        ESCREXDADOS(A,B,C,LINXSEG,PRIXLIN,PRIXCOL,
                                    TERRA);
                        IF REDUC = 0 AND EMPAREL ¬= 5
                        THEN
                        BEGIN
                           RECLCOES(A,C,PARTICI,PRIXLIN,PRIXCOL,
                                    RECOBRE,TERRA,X);
                           REA_XPONT(LINXSEG,PRIXLIN,TERRA)
                        END;
                        IF EMPAREL = 5
                        THEN
                          BEGIN
                             FOR I:=1 STEP 1 UNTIL M
                             DO
                             B[I]:=-B[I];
```

```
                CSTXNEG:=0;
                FOR J:=1 STEP 1 UNTIL N
                DO
                BEGIN
                   CSTXNEG:= * + C[J];
                   FOR I:=1 STEP 1 UNTIL M
                   DO
                   B[I]:=B[I]+A[I,J]
                END
              END;
            ENUXIMPXRECXPAR(A,B,C,INDICE,LINXSEG,
                            PRIXLIN,PRIXCOL,SOMA,
                            TEMXSOLUCAO,TERRA,X,
                            XSOLXATUAL,Y,ZA,ZERO,
                            ZXMENOR,ZXOTIMO);
            IF RECOBRE = 2
            THEN
              WRITE(SAIDA,<//,X10,
"RESULTADOS PARA O PROBLEMA DE DETERMINAR C CONJUNTO RECO-",//,
X10,"BRIMENTO MINIMO PROPOSTO:",//>);
            IF RECXGRA = 4
            THEN
              WRITE(SAIDA,<//,X10,
"RESULTADOS PARA O PROBLEMA DE DETERMINAR O RECOBRIMENTO ",//,
X10,"MINIMO DOS NOS PELOS ARCOS DE UM GRAFO NAO ORIENTADO,",//,
X10,"PROPOSTO",//>);
            IF RECOBRE = 2 OR RECXGRA = 4
            THEN
              ESCREXRESUL(C,TEMXSOLUCAO,X,XSOLXATUAL,
                          ZA,ZXOTIMO);
            IF EMPAREL = 5
            THEN
              BEGIN
                 ZA:=0;
                 FOR J:=1 STEP 1 UNTIL N
                 DO
```

```
                        BEGIN
                           X[J]:=XSOLXATUAL[J];
                           ZA:= * + C[J]*X[J];
                           X[J]:=1-X[J]
                        END;
                        ZXOTIMO:=ZA+CSTXNEG;
                        ZXOTIMO:=-ZXOTIMO;
                        WRITE(SAIDA,<//,X10,
"RESULTADOS PARA O PROBLEMA DE DETERMINAR O EMPARELHAMENTO",//,
X10,"MAXIMO DOS NOS PELOS ARCOS DE UM GRAFO NAO ORIENTADO ,",//,
X10,"PROPOSTO:",//>);
                        IF TEMXSOLUCAO
                        THEN
                          BEGIN
                             WRITE(SAIDA,<//,X10,
"INDICE DAS VARIAVEIS IGUAIS A 1 NA SOLUCAO",//>);
                             WRITE(SAIDA,<//,X1,25I5,/>,FOR
                                   J:=1 STEP 1 UNTIL N DO
                                   IF X[J] = 1 THEN J);
                             WRITE(SAIDA,<//,X10,
                                   "VETOR SOLUCAO",//>);
                             WRITE(SAIDA,<//,X1,25I5,/>,FOR
                                   J:=1 STEP 1 UNTIL N DO X[J]);
                             WRITE(SAIDA,<//,X10,
"VALOR OTIMO DA FUNCAO OBJETIVO",I10,//,130("="),//>,ZXOTIMO)
                          END
                        ELSE
                          WRITE(SAIDA,<//,X10,
"O PROBLEMA  DE DETERMINAR O EMPARELHAMENTO MAXIMO DOS NOS",//,
X10,"PELOS ARCOS DE UM GRAFO NAO ORIENTADO, NAO TEM SOLUCAO",//>
                                                              );
                     END;
                  END
               END;
            IF PARTICI = 3
            THEN
```

```
            BEGIN
               MK:=M;
               M:=2*M
            END;
         IF PARTICI = 3
         THEN
            BEGIN
               INTEGER DISPO,I,J,LINHA,CCLUNA,PONTEIRO,TERRA,
                       ZA,ZXMENOR,ZXOTIMO;
               INTEGER ARRAY A[0:4,1:ENTIER(M*N/2)],B[1:M],
                             C[1:N],INDICE[1:N],LINXSEG[0:M],
                             PRIXLIN[1:M],PRIXCOL[1:N],
                             SOMA[1:N],X[1:N],Y[1:M],
                             XSOLXATUAL[1:N],ZERO[1:N];
               BOOLEAN TEMXSOLUCAO;
               AJUSXPONT(A,LINXSEG,PRIXLIN,PRIXCOL,TERRA,X);
%--------------LEITURA DOS CUSTOS
               READ(ENTRADA,<16I5>,FOR J:=1 STEP 1 UNTIL N
                                   DO C[J]);
               DISPO:=1;
%--------------LEITURA DA MATRIZ A
               READ(ENTRADA,<2I4>,LINHA,COLUNA);
               WHILE LINHA ¬= 0
               DO
               BEGIN
                  PONTEIRO:=DISPO;
                  A[0,PONTEIRO]:=1;
                  A[1,PONTEIRO]:=LINHA;
                  A[2,PONTEIRO]:=COLUNA;
                  DISPO:=A[4,DISPO];
                  ATIVXPONT(A,LINHA,COLUNA,PONTEIRO,PRIXLIN,
                            PRIXCOL,TERRA,X);
                  READ(ENTRADA,<2I4>,LINHA,CCLUNA)
               END;
%--------------LEITURA DO VETOR B
               READ(ENTRADA,<16I5>,FOR I:=1 STEP 1 UNTIL MK
```

```
                                DO B[I]);
            ESCREXDADOS(A,B,C,LINXSEG,PRIXLIN,PRIXCOL,
                        TERRA);
            IF REDUC = 0
            THEN
            BEGIN
               REDUCOES(A,C,PARTICI,PRIXLIN,PRIXCOL,
                        RECCBRE, TERRA,X);
               REAJXPONT(LINXSEG,PRIXLIN,TERRA)
            END;
            FOR I:=1 STEP 1 UNTIL MK
            DO
            BEGIN
               INTEGER KK;
               KK:=PRIXLIN[I];
               WHILE KK ¬= TERRA AND DISPO ¬= TERRA
               DO
               BEGIN
                  PONTEIRO:=DISPO;
                  A[0,PONTEIRO]:=-A[0,KK];
                  A[1,PONTEIRO]:=LINHA:=MK+A[1,KK];
                  A[2,PONTEIRO]:=COLUNA:=A[2,KK];
                  B[MK+A[1,KK]]:=-B[A[1,KK]];
                  KK:=A[4,KK];
                  DISPO:=A[4,DISPO];
                  ATIVXPONT(A,LINHA,COLUNA,PONTEIRO,
                            PRIXLIN,PRIXCOL,TERRA,X)
               END;
               IF DISPO = TERRA
               THEN
               WRITE(SAIDA,<//,X10,
"FORAM ESGOTADAS AS POSICOES DA MATRIZ. PROVAVELMENTE ELA",//,
X10,"NAO E ESPARSA. TENTE UTILIZAR A PROCECURE ENUMXIMPL.",//>)
            END;
            IF DISPO ¬= TERRA
            THEN
```

```
                    ENUXIMPXRECXPAR(A,B,C,INDICE,LINXSEG,
                                    PRIXLIN,PRIXCOL,SOMA,
                                    TEMXSOLUCAO,TERRA,X,
                                    XSOLXATUAL,Y,ZA,ZERO,
                                    ZXMENOR,ZXOTIMO);
                  WRITE(SAIDA,<//,X10,
 "RESULTADOS PARA O PROBLEMA DE DETERMINAR O CONJUNTO",//,
 X10,"PARTICIONAMENTO MINIMO PROPOSTO:",//>);
                  ESCREXRESUL(C,TEMXSOLUCAO,X,XSOLXATUAL,ZA,
                              ZXOTIMO);
%-----------------PODE SER IMPRESSO O RETRATO DA MEMORIA OU
%-----------------DISPOSICAO DOS DADOS NA MEMORIA
               END
          END;
%-------------------------------------------------------------%
   READ(ENTRADA,<2I3>,M,N);
   WHILE M <> 0
   DO
   BEGIN
      ENTRADAXSAIDA;
      READ(ENTRADA,<2I3>,M,N)
   END
%-------------------------------------------------------------%
END.
```

SOBRE OS DADOS PARA TESTE DO PROGRAMA

Transcrevemos aqui os dados utilizados para testar o programa que busca a solução de um problema de Programação Inteira bivalente, pelo método de Enumeração Implícita de pseudo-soluções parciais. Estes dados foram obtidos de PETERSEN[47]. A solução ótima deste problema, sem a utilização de filtros, não foi obtída em 5 horas de processamento no B/6700, porém com a implementação dos filtros, ela foi alcançada em 19 minutos.

Custos:

560	1125	300	620	2100	431	68	328	47	122	322	196	41	25	425
4260	416	115	82	22	631	132	420	86	42	102	215	81	91	26
49	420	316	72	71	49	108	116	90	738	1811	430	3060	215	58
296	620	418	47	81										

Matriz A(I,J):

40	91	10	30	160	20	3	12	3	18	9	25	1	1	10
280	10	8	1	1	49	8	21	6	1	5	10	8	2	1
0	10	42	6	4	8	0	10	1	40	86	11	120	8	3
32	28	13	2	4										
16	92	41	16	150	23	4	18	6	0	12	8	2	1	0
200	20	6	2	1	70	9	22	4	1	5	10	6	4	0
4	12	8	4	3	0	10	0	6	28	93	9	30	22	0
36	45	13	2	2										

38	39	32	71	80	26	5	40	8	12	30	15	0	1	23
100	0	20	3	0	40	6	8	0	6	4	22	4	6	1
5	14	8	2	8	0	20	0	0	6	12	6	80	13	6
22	14	0	1	2										

8	71	30	60	200	18	6	30	4	8	31	6	3	0	18
60	21	4	0	2	32	15	31	2	2	7	8	2	8	0
2	8	6	7	1	0	0	20	8	14	20	2	40	6	1
14	20	12	0	1										

38	52	30	42	170	9	7	20	0	3	21	4	1	2	14
310	8	4	6	1	18	15	38	10	4	8	6	0	0	3
0	10	6	1	3	0	3	5	4	0	30	12	16	18	3
16	22	30	4	0										

Vetor B:

800	650	550	550	650

Solução:

X(I) = 1 para I = 4, 6, 8, 9, 11, 12, 13, 15, 16, 17, 19, 20, 23, 25, 26, 27, 28, 29, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 47, 48, 49, 50.

Valor ótimo da função objetivo: 16537.

Transcrevemos também os dados correspondentes ao problema gerado para teste da parte do programa que determina o conjunto recobrimento ou particionamento mínimo. Ele corresponde a uma parte de um problema mais amplo que trata da alocação ótima de tripulações em rotas aéreas. O trecho que testamos é exatamente o que faz a busca do ótimo após a geração da matriz das rotações. Neste problema temos 16 vôos e 75 rotações geradas com esses vôos. Assim, trata-se de um problema de 75 variáveis e 16 restrições.

Custos:

50	60	120	70	160	110	70	75	65	65	45	125	105	75	70
70	210	205	190	150	145	155	190	150	220	100	150	105	105	110
115	170	175	160	170	180	125	175	185	180	190	125	165	120	130
110	120	115	150	95	100	205	215	185	95	90	145	90	175	170
180	155	150	160	185	175	115	125	120	115	160	120	120	115	125

Vetor B:

$B(I) = 1$, para $I = 1, \ldots, 16$.

Matriz A(I,J):

$A(I,J) = 1$ para $I = 1$, $J = 1, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26$

$I = 2$, $J = 2, 19, 20, 21, 22, 27, 28, 29, 30, 31$

$I = 3$, $J = 3, 17, 18, 32, 33$

I = 4, J = 4, 34, 35, 36, 37

I = 5, J = 5, 23, 54

I = 6, J = 6, 19, 27, 34, 43, 71

I = 7, J = 7, 17, 20, 30, 32, 42, 48, 55, 59, 62, 72, 73

I = 8, J = 8, 38, 39, 40, 41, 42, 43, 44, 45

I = 9, J = 9, 24, 28, 35, 36, 38, 39, 46, 47, 48, 56, 57, 70

I = 10, J = 10, 18, 21, 24, 29, 35, 38, 40, 44, 46, 52, 57, 58, 60, 63, 66, 67, 74

I = 11, J = 11, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58

I = 12, J = 12, 25, 49, 52, 53, 59, 60, 61

I = 13, J = 13, 62, 63, 64

I = 14, J = 14, 22, 25, 31, 33, 36, 37, 39, 41, 45, 47, 51, 53, 61, 64, 65, 68, 75

I = 15, J = 15, 65, 66, 67, 68, 69, 70, 71, 72

I = 16, J = 16, 26, 40, 41, 50, 65, 66, 69, 73, 74, 75

BIBLIOGRAFIA

BIBLIOGRAFIA

1. ARABEYRE, J. P., FEARNLEY, J., STEIGER, F. C., TEATHER, W. - The Airline Crew Scheduling Problem: A Survey, Transportation Science, vol. 3, 1969, pp. 140-163.

2. BALAS, E. - An Additive Algorithm for Solving Linear Programs with Zero-One Variables, Operations Research, vol. 13, 1965, pp. 517-546.

3. BALAS, E. - Discrete Programming by the Filter Method, Operations Research, vol. 15, 1967, pp. 915-957.

4. BALAS, E., PADBERG, M. W. - On the Set Covering Problem, Carnegie-Mellon University, Management Science Research Report nº 197, 1970.

5. BALAS, E. - Bivalent Programming by Implicit Enumeration, Encyclopedia of Computer Science and Technology, vol. 2, 1975, pp. 479-494.

6. BALAS, E., PADBERG, M. W. - Set Partitioning, Carnegie-Mellon University, Combinatorial Programming: Methods and Applications, 1975, pp. 205-258.

7. BALAS, E., PADBERG, M. W. - Set Partitioning: A Survey, Siam Review, nº 4, 1976, pp. 710-760.

8. BALAS, E., SAMUELSSON, H. - A Node Covering Algorithm, Naval Research Logistics Quarterly, vol. 24, 1977, pp. 213-233.

9. BALINSKI, M. L. - Integer Programming: Methods, Uses, Computation, Management Science, vol. 12, 1965, pp. 253-313.

10. BALINSKI, M. L. - On Maximum Matching, Minimum Covering and their Connections, Proceedings of the Princeton Symposium on Mathematical Programming, Princeton University Press, 1970, pp. 303-311.

11. BERGE, C. - Graphs and Hypergraphs, North-Holland/American Elsevier, 1973.

12. BERZTISS, A. T. - Data Structures Theory and Practice, Academic Press, 1971.

13. BYRNE, J. L., PROLL, L. G. - Solution of Linear Programming in 0-1 Variables by Implicit Enumeration, Comm. ACM, vol. 11, 1968, pp. 782.

14. DAHL, O. J., DIJKSTRA, E. W., HOARE, C. A. R. - Structured Programming, Academic Press, 1972.

15. DANTZIG, G. B. - Linear Programming and Extensions, Princeton University Press, 1963.

16. DELORME, M. J. - Contribution à la Résolution du Problèmme de Partitionnement: Methodes de Troncatures, These de Docteur Ingenieur Présentée à L'Universite de Paris, 1974.

17. DELORME, J., HEURGON, E. - Problemes de Partitionnement: Exploration Arborescente ou Methode de Troncatures?, Revue Française d'Automatique, Informatique et Recherche Opérationnelle, vol. 2, 1975, pp.53-65.

18. FIALA, F. - Solution of Linear Programming Problems in 0-1 Variables, Comm. ACM, vol. 16, 1973, pp. 445-447.

19. FORD, L. R., FULKERSON, D. R. - Flows in Network, Princeton University Press, 1962.

20. GARFINKEL, R. S., NEMHAUSER, G. L. - The Set-Partitioning Problem: Set-Covering with Equality Constraints, Operations Research, vol. 17, 1969. pp. 848-856.

21. GARFINKEL, R. S., NEMHAUSER, G. L. - Integer Programming, John Wiley & Sons, 1972.

22. GEOFFRION, A. M. - Integer Programming by Implicit Enumeration and Balas Method, Siam Review, vol. 9, nº 2, 1967, pp. 178-190.

23. GEOFFRION, A. M. - An Improved Implicit Enumeration Approach for Integer Programming, Operations Research, vol. 17, 1969, pp. 437-454.

24. GLOVER, F., ZIONTS, S. - A Note on the Additive Algorithm of Balas, Operations Research, vol. 13, 1965, pp. 546-549.

25. GLOVER, F. - A Multiphase-dual Algorithm for the Zero-One Integer Programming Problem, Operations Research, vol. 13, 1965, pp. 879-919.

26. GLOVER, F. - Surrogate Constraints, Operations Research, vol. 16, 1968, pp. 741-749.

27. GLOVER, F. - A Note on Extreme-Point Solutions and a Paper by Lemke, Salkin, and Spielberg, Operations Research, Vol. 19, 1971, pp. 1023-1025.

28. GLOVER, F. - Surrogate Constraint Duality in Mathematical Programming, Operations Research, vol. 23, 1975, pp. 434-451.

29. GONDRAN, M., LAURIÈRE, J. L. - Un Algorithme pour le Probleme de Partitionnement, Revue Française d'Automatique, Informatique et Recherche Opérationnelle, vol. 1, 1974, pp. 27-40.

30. GONDRAN, M., LAURIÈRE, J. L. - Un Algorithme pour les Problemes de Recouvrement, Revue Française d'Automatique, Informatique et Recherche Opérationnelle, vol. 2, 1975, pp. 33-51.

31. GREENBERG, H. - Integer Programming, Academic Press, 1971.

32. GUE, R. L., LIGGETT, J. C. - Analysis of Algorithms for the Zero-One Programming Problem, Comm. ACM, vol. 11, 1968.

33. GUHA, D. - The Set-covering Problem with Equality Constraint, Operations Research, vol. 21, 1973, pp. 348-351.

34. HEURGON, E. - Un Probleme de Recouvrement: L'Habillage des Horaires d'une Ligne d'Autobus, Revue Française d'Automatique, Informatique et Recherche Opérationnelle, vol. 1, 1971, pp. 13-29.

35. KNUTH, D. E. - The Art of Computer Programming, Addison - Wesley Publishing Company, 1973.

36. KOVÂCS, L. B. - A New Solution for the General Set Covering Problem, Lecture Notes in Computer Science, 5th Conference on Optimization Techniques, Part I, Springer Verlag, 1973.

37. KÜNZI, H. P., TZSCHACH, H. G., ZEHNDER, C. A. - Numerical Methods of Mathematical Optimization, Academic Press, 1971.

38. LAND, A. H., DOIG, A. G. - An Automatic Method of Solving Discrete Programming Problems, Econometrica, vol. 28, 1960, pp. 497-520.

39. LEMKE, C. E., SPIELBERG, K. - Direct Search Algorithm for Zero-One and Mixed-Integer Programming, Operations Research, vol. 15, 1967, pp. 892-914.

40. LEMKE, C. E., SALKIN, H. M., SPIELBERG, K. - Set-Covering by Single-branch Enumeration with Linear-programming Subproblems, Operations Research, vol. 19, 1971, pp. 998-1022.

41. MACULAN, N. - Programação Linear Inteira, Notas preparatórias de um livro texto, cap. V, COPPE/IMUFRJ, 1977.

42. MAHENDRARAJAH, A., FIALA, F. - A Comparison of three Algorithms for Linear Zero-One Programs, ACM Transactions on Mathematical Software, vol. 2, 1976, pp. 331-334.

43. MURTY, K. G. - Linear and Combinatorial Programming, John Wiley & Sons, Inc., 1976.

44. NICOLETTI, B. - Automatic Crew Rostering, Transportation Science, vol. 9, 1975, pp. 33-42.

45. ORGANICK, E. I. - Computer System Organization, The B5700/B6700 Series, Academic Press, Inc., 1973.

46. PAGE, E. S., WILSON, L. B. - Information Representation and Manipulation in a Computer, Cambridge University Press, 1973.

47. PETERSEN, C. - Computational Experience with Variants of the Balas Algorithm Applied to Selection of R&D Projects, Management Science, vol. 13, 1967, pp. 736-750.

48. PIPER, J. C., ZOLTNERS, A. A. - Implicit Enumeration Based Algorithms for Postoptimizing Zero-One Programs, Naval Research Logistics Quarterly, vol. 22, 1975, pp. 791-809.

49. RICHARDSON, R. - An Optimization Approach to Routing Aircraft, Transportation Science, vol. 10, 1976, pp. 52-71.

50. ROODMAN, G. M. - Postoptimality Analysis in Zero-One Programming by Implicit Enumeration, Naval Research Logistics Quarterly, vol. 19, 1972, pp. 435-447.

51. ROTH, R. - Computer Solutions to Minimum-Cover Problems, Operations Research, vol. 17, 1969, pp. 455-466.

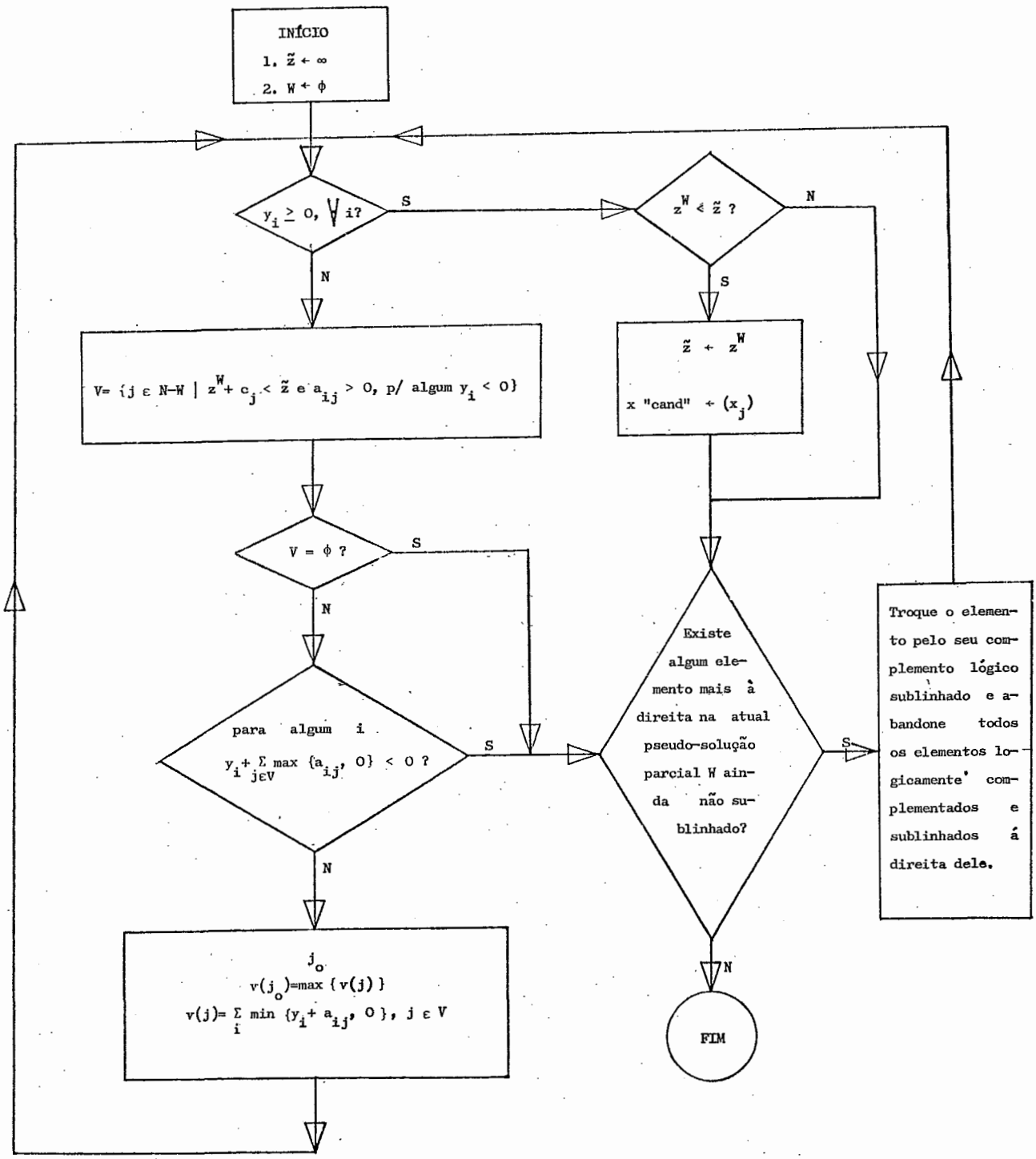
52. SALKIN, H. M. - Integer Programming, Addison-Wesley Publishing Company, 1975.

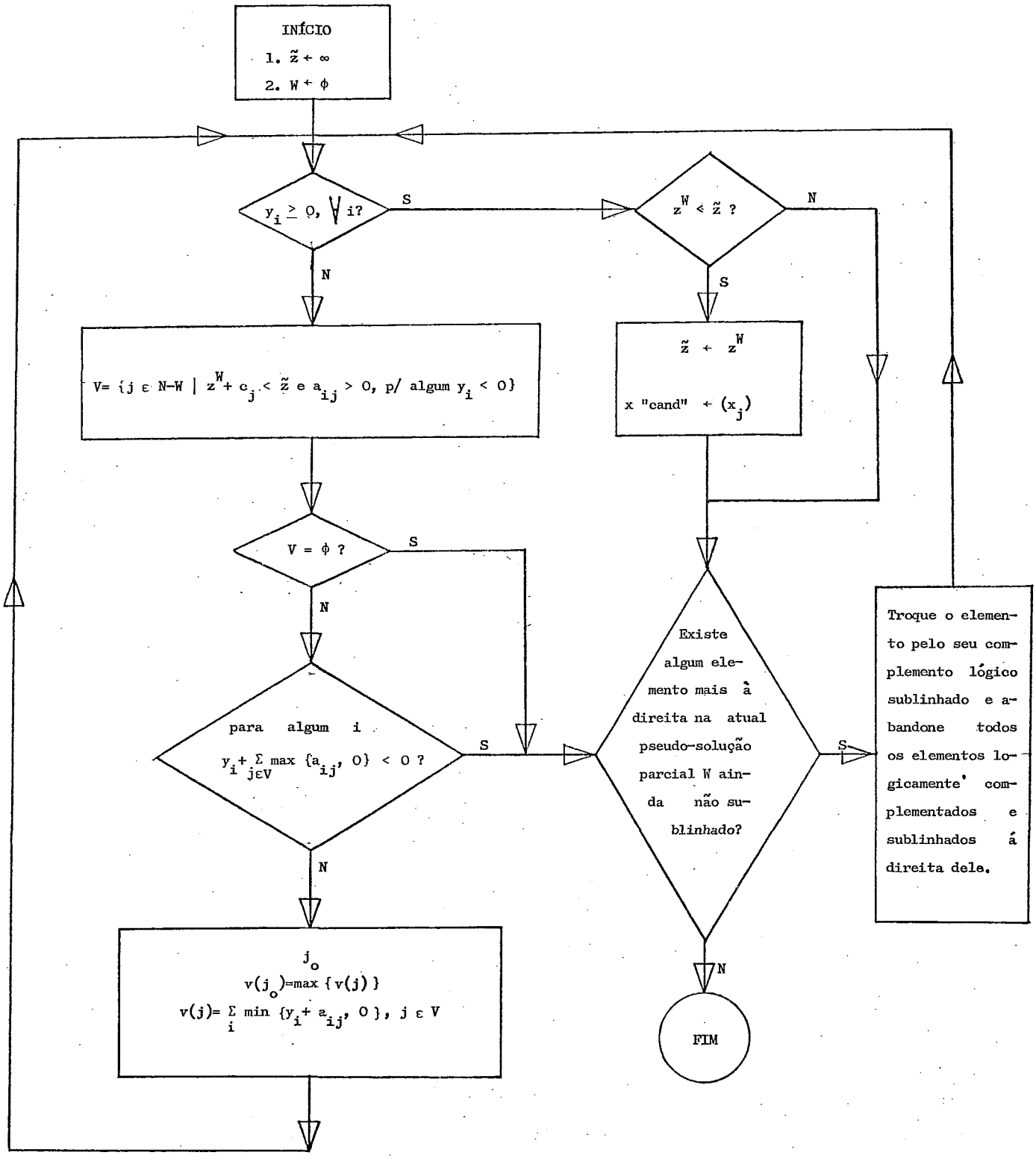
53. TAHA, H. A. - Integer Programming Theory, Applications, and Computations, Academic Press, 1975.

54. THE BURROUGHS CORP. - B6700/B7700 ALGOL Language Reference Manual, 1974.

55. TUAN, N. P. - A Flexible Tree-search Method for Integer Programming Problems, Operations Research, vol. 19, 1971, pp. 115-119.

56. ZIONTS, S. - Linear and Integer Programming, Prentice Hall, Inc., 1974.


INÍCIO
1. $\tilde{z} \leftarrow \infty$
2. $W \leftarrow \phi$
$y_i \geq 0, \forall i$?
S
N
$z^W < \tilde{z}$?
N
S
$V = \{j \in N-W \mid z^W + c_j < \tilde{z} \text{ e } a_{ij} > 0, \text{ p/ algum } y_i < 0\}$
$\tilde{z} \leftarrow z^W$
x "cand" $\leftarrow (x_j)$
$V = \phi$?
S
N
para algum i
$y_i + \sum_{j \in V} \max \{a_{ij}, 0\} < 0$?
S
N
Existe algum elemento mais à direita na atual pseudo-solução parcial W ainda não sublinhado?
S
N
Troque o elemento pelo seu complemento lógico sublinhado e abandone todos os elementos logicamente complementados e sublinhados á direita dele.
j_o
$v(j_o) = \max \{v(j)\}$
$v(j) = \sum_i \min \{y_i + a_{ij}, 0\}, j \in V$
FIM


INÍCIO
1. $\tilde{z} \leftarrow \infty$
2. $W \leftarrow \phi$
$y_i \geq 0, \forall i$?
S
N
$z^W < \tilde{z}$?
N
S
$V = \{j \in N-W \mid z^W + c_j < \tilde{z} \text{ e } a_{ij} > 0, \text{ p/ algum } y_i < 0\}$
$\tilde{z} \leftarrow z^W$
x "cand" $\leftarrow (x_j)$
$V = \phi$?
S
N
para algum i
$y_i + \sum_{j \in V} \max \{a_{ij}, 0\} < 0$?
S
N
Existe algum elemento mais à direita na atual pseudo-solução parcial W ainda não sublinhado?
S
N
Troque o elemento pelo seu complemento lógico sublinhado e abandone todos os elementos logicamente complementados e sublinhados á direita dele.
j_o
$v(j_o) = \max \{v(j)\}$
$v(j) = \sum_i \min \{y_i + a_{ij}, 0\}, \; j \in V$
FIM