TÍTULO DA TESE

O PROBLEMA DO CORTE BIDIMENSIONAL

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida por **Maria do Socorro Nogueira Rangel** e aprovada pela comissão julgadora.

Campinas, de de 1990

Prof. Dr. _____

Clovis Perin Filho

ORIENTADOR

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requesito parcial para obtenção do Título de Mestre em **Matemática Aplicada**.

Aos meus pais.

Ao Anísio - amor perto/longe.

Aos meus irmãos: Antônia, Rosa, Manoel, Luísa, Pedro e Mário.

Agradecimentos:

- Ao Clovis, pela dedicação e paciência;

- Às companheiras de morada - Rita e Osima; Siomara, Karla e Cassiana; Adriana; Simone; Rita - pela compreenção, carinho ... amizade;

- Ao Chico, Jonatas, Zé Augusto, Vinícius , Jurandir - Turma do Mestrado 1986 - Pelas trocas de idéias e amizade;

- Aos colegas de mestrado - Inês, Rose, Denise, Regina, Cecília, Andréa, Emérito, Júlia e Adriana;

- Aos colegas de trabalho Lúcia, Geraldo e Heloísa;

- Aos professores - Boldrini e Martínez (UNICAMP); Valdir, Tarcísio, Shirley e Genesio (UFGo) - pelo estímulo e exemplo;

- Ao CNPQ, FAPESP e UNICAMP pelo apoio financeiro.

CONTEÚDO

CAPÍTULO V

INTRODUÇÃO

Em diversas indústrias, um determinado material é produzido em peças de tamanho padrão pré-fixados. Em geral a encomenda por este material é feita para peças de tamanhos menores. Trata-se de um importante problema de otimização denominado Problema do Corte. Ou seja, o problema de determinar esquemas de corte para dividir a peça padrão em peças menores, de tal forma que o custo de produção seja minimizado. Se apenas a largura (ou o comprimento) das peças é considerada no problema do corte, tem-se o Problema do Corte Unidimensional, que ocorre principalmente nas indústrias de papel, barras de ferro, papelão, etc. O Problema do Corte Bidimensional está presente quando duas dimensões (por exemplo comprimento e largura) são consideradas e tem aplicações nas indústrias de vidro, aço, móveis, roupas, diagramação de jornais, etc.

O objetivo do presente trabalho é fazer um estudo do Problema do Corte Bidimensional, assim como desenvolver e implementar algoritmos para resolvê-lo. Para realizar este estudo é necessário o conhecimento de conceitos e técnicas de resolução de problemas relacionados tais como: Problema da Mochila (Kanapsack Problem) e suas variações, e o Problema do Corte Unidimensional.

No capítulo I são resumidos os resultados da Programação Linear usados no desenvolvimento dos capítulos seguintes. O capítulo II é dedicado ao estudo do Problema da Mochila. Nos capítulos III e IV são estudados os Problemas do Corte

Unidimensional e Bidimensional, respectivamente. Finalmente no capítulo V são apresentados os resultados computacionais da implementação de algoritmos para resolver o Problema do Corte Bidimensional.

CAPÍTULO I

1 - PROGRAMAÇÃO LINEAR

Os diversos problemas de Programação Matemática tratados neste trabalho são estudados com o auxílio da Programação Linear. Estes problemas possuem características próprias tais como: i) número de variáveis muito superior ao número de restrições; ii) colunas da matriz de restrições composta por números inteiros positivos; iii) variáveis e restrições canalizadas. As características i) e ii) indicam o uso do método Simplex conjugado a técnicas de geração de colunas, enquanto iii) indica o uso de técnicas de canalização. Nas próximas seções são resumidos alguns resultados da Programação Linear e descritas adaptações do método Simplex para trabalhar com as características acima.

Em 1984, Karmarkar [K], prôpos um método projetivo para resolver programas lineares que tem se mostrado mais eficiente do que o método Simplex para problemas de grande porte. Entretanto, nos problemas tratados no presente trabalho, apenas uma parte da matriz de restrições é armazenada, ou seja, não se dispõe do interior da região factível e portanto não é possível resolvê-los através dos métodos de ponto interior.

1.1 - FORMULAÇÃO E ALGUNS RESULTADOS

Um problema de Programação Linear padrão consiste em determinar x (vetor n x 1) tal que:

$$\begin{aligned} P: \quad & \min \quad cx \\ & s.a \quad Ax = b \\ & \qquad x \geq 0 \end{aligned} \tag{1.1}$$

onde A : matriz m x n com posto m

c : vetor 1 x n

b : vetor m x 1

Associado a cada Problema de Programação Linear, existe um outro problema chamado Problema Dual. Se um problema Primal é da forma (1.1) então o problema dual correspondente consiste em determinar π (vetor m x 1) tal que:

$$\begin{aligned} D: \quad & \max \quad \pi b \\ & s.a \quad \pi A \leq c \\ & \qquad \pi \text{ irrestrito} \end{aligned}$$

Um vetor x é primal factível se $Ax = b$ e $x \geq 0$; uma solução primal factível x^* é ótima se $cx^* \leq cx$ para todo x factível. Uma base B é uma submatriz não singular de A com m colunas. Supondo, sem perda de generalidade, que é feita uma reordenação nas colunas de A convertendo-a em [B | N], então as soluções básicas associadas a B são expressas por : $x_B = B^{-1}b$, $x_N = 0$ e $\pi = c_B B^{-1}$.

Abaixo estão descritos os principais resultados relacionando os problemas primal e dual. [Ba]

TEOREMA FRACO DA DUALIDADE - Se $\bar{x}$ e $\bar{\pi}$ são duas soluções factíveis para os problemas P e D respectivamente, então:

$$c\bar{x} \geq \bar{\pi}b.$$

COROLÁRIO DO TEOREMA FRACO - Se x^* e π^* são soluções factíveis com:

$$cx^* = \pi^*b$$

então são soluções ótimas.

TEOREMA FUNDAMENTAL DA DUALIDADE - Se um dos problemas P ou D possui solução ótima então o outro problema também possui solução ótima e o valor da função objetivo é o mesmo.

TEOREMA DAS FOLGAS COMPLEMENTARES - As soluções factíveis x^* e π^* são ótimas se e somente se satisfazem as condições de folga complementar:

$$(c_j - \pi^* A_j)\, x_j^* = 0 \qquad j = 1..n$$

onde A_j é a coluna j da matriz A.

1.2 - MÉTODOS DE SOLUÇÃO

1.2.1 - MÉTODO SIMPLEX REVISADO [Mu]

1) INICIALIZAÇÃO - Seja o problema (1.1). Suponha que uma solução básica factível (SBF) inicial associada a uma base B seja conhecida. Tal solução, se existe, pode ser encontrada através do método das penalidades (grande M) ou das duas fases .

2) OTIMALIDADE - Para a melhorar a SBF atual, é necessário encontrar uma variável não-básica com custo relativo satisfazendo:

$$\bar{c}_s = c_s - \pi A_s < 0 \qquad (1.2)$$

onde π é solução do sistema :

$$\pi B = c_B \qquad (1.3)$$

Se não existe variável não-básica satisfazendo (1.2), pare; a solução atual é ótima.

3) TESTE DA RAZÃO - A escolha da coluna que irá deixar a base é feita primeiro resolvendo o sistema :

$$B y = A_s \qquad (1.4)$$

Se $y \leq 0$, pare; a solução é ilimitada. Caso contrário, encontre um índice r que satisfaz

$$\bar{b}_r / y_{rs} = \min_i \{ \bar{b}_i / y_{is} , y_{is} > 0 \}.$$

onde: $\bar{b} = x_B$

4) ATUALIZAÇÃO - A atualização é feita, retirando a coluna $A_{B(r)}$ da base e colocando em seu lugar a coluna A_s. A nova solução corrente é obtida resolvendo o sistema :

$$B x_B = b \qquad (1.5)$$

Volte para 2).

Os sistemas de equações lineares (1.3), (1.4), (1.5) podem ser resolvidos através da matriz B^{-1} explícita ou escrita como um produto de matrizes. O algoritmo simplex implementado neste trabalho resolve os sistemas lineares através da fatoração de Cholesky da matriz BB^t .

FATORAÇÃO DE CHOLESKY

Seja B uma matriz básica. BB^t é uma matriz positiva definida e portanto pode ser decomposta em um produto de matrizes da forma: $BB^t = LL^t$, onde L é triângular inferior. No presente trabalho L é chamada fator de Cholesky da matriz BB^t associado à base B.

Uma matriz quadrada Q é uma matriz ortogonal, se $QQ^t = I$. Como B é uma matriz não singular, então existe uma matriz ortogonal Q tal que

$$L = BQ$$

pois

$$LL^t = BQQ^tB^t = BB^t.$$

Seja um vetor A_k com componente $a_{jk} \neq 0$. Se para algum $i \neq j$ existe um elemento $a_{ik} \neq 0$, então ele pode ser reduzido a zero pré-multiplicando o vetor A_k por uma matriz de rotação de Givens. Isto é:

$$(P_i^j A_k)^t = (a_{1k} \ldots a_{(j-1)k}, \gamma, a_{(j+1)k} \ldots a_{(i-1)k}, 0, a_{(i+1)k} \ldots a_{km})$$

onde:

$$P_i^j = \begin{bmatrix} & 0 & & 0 & \\ I & \vdots & 0 & \vdots & 0 \\ & 0 & & 0 & \\ 0\ldots0 & C & 0\ldots0 & S & 0\ldots0 \\ & 0 & & 0 & \\ 0 & \vdots & I & \vdots & 0 \\ & 0 & & 0 & \\ 0\ldots0 & -S & 0\ldots0 & C & 0\ldots0 \\ & 0 & & 0 & \\ 0 & \vdots & 0 & \vdots & I \\ & 0 & & 0 & \end{bmatrix}$$

e:

I = matriz identidade, 0 = matriz nula, $\gamma = \sqrt{a_{jk}^2 + a_{ik}^2}$,

$C = a_{jk} / \gamma$, $S = a_{ik} / \gamma$

Assim, para obter o fator de Cholesky L associado a uma dada base B pode se utilizar rotações de givens, isto é, pré-multiplicar a matriz B^t por P_i^1, i = 2..m, transformando os elementos da coluna 1 abaixo da diagonal em zero. Em seguida pré-multiplicar a matriz resultante por uma sequência de matrizes P_i^2, i = 3..m, transformando os elementos da coluna 2 abaixo da diagonal em zero. O processo é repetido m-3 vezes obtendo como resultado uma matriz triângular superior. Finalmente a transposta desta matriz é o fator de Cholesky associado à matriz B.

Resolver um sistema de equações na forma:

$$By = b$$

usando fatoração de Cholesky da Matriz BB^t, consiste em primeiro resolver em v, por substituição, o sistema triângular:

$$Lv = b$$

e então encontrar o vetor u resolvendo, também por substituição, mais um sistema triângular:

$$L^t u = v.$$

A solução do sistema By = b é : $y = B^t u$, pois:

$$Lv = b \rightarrow LL^t u = b \rightarrow BB^t u = b \rightarrow By = b.$$

O mesmo raciocínio é válido para um sistema na forma : $\pi B = c$.

A cada iteração do simplex temos uma mudança na matriz básica B. Refatorar a matriz básica a cada iteração é um esforço computacional que pode ser evitado fazendo apenas a atualização do fator de Cholesky associado á base B.

ATUALIZAÇÃO DO FATOR DE CHOLESKY

Seja B a presente base e L o fator de Cholesky associado. Suponha que A_s é a coluna que vai entrar na base e A_r a coluna que vai sair. Se $\hat{B}$ é a nova base, então:

$$\hat{B}\hat{B}^t = BB^t - A_r A_r^t + A_s A_s^t$$

onde cada uma das matrizes $A_r A_r^t$ e $A_s A_s^t$ tem posto 1. O fator de Cholesky associado à nova base é obtido em duas etapas:

<u>1ª etapa</u> - É obtido o fator de Cholesky da matriz $BB^t - A_r A_r^t$. Seja $R = L^t$. Encontre o vetor p que resolve o sistema:

$$R^t p = A_r . \qquad (1.6)$$

Construa uma matriz $F_{(m+1)x(m+1)}$ na forma:

$$F = \left[\begin{array}{c|c} 0 & 0 \ \ldots \ 0 \\ \hline p & R \end{array}\right]$$

Pré-multiplique F por matrizes de rotações de Givens P_i^1 , i = m+1, m..2. Isto anula os elementos da coluna 1 de F sem alterar a estrutura triângular superior de R. A matriz resultante é:

$$\left[\begin{array}{c|c} \gamma & v \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & \bar{R} \end{array}\right]$$

onde: $\gamma = 1$, $v = A_r^t$. $\bar{R}^{\,t}$ é fator de Cholesky de $BB^t - A_r^t A_r$ pois:

$$\bar{Q}\left[\begin{array}{c|c} 0 & 0 \ \ldots \ 0 \\ \hline p & R \end{array}\right] = \left[\begin{array}{c|c} \gamma & v \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & \bar{R} \end{array}\right]$$

$$\left(\bar{Q}\left[\begin{array}{c|c} 0 & 0 \ \ldots \ 0 \\ \hline p & R \end{array}\right]\right)^t \bar{Q}\left[\begin{array}{c|c} 0 & 0 \ \ldots \ 0 \\ \hline p & R \end{array}\right] = \left(\bar{Q}\left[\begin{array}{c|c} 0 & 0 \ \ldots \ 0 \\ \hline p & R \end{array}\right]\right)^t \left[\begin{array}{c|c} \gamma & v \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & \bar{R} \end{array}\right]$$

mas, $\bar{Q}^t\bar{Q}=I$, então:

$$\left[\begin{array}{c|c} p^t p & p^t R \\ \hline R & R^t R \end{array}\right] = \left[\begin{array}{c|c} \gamma^2 & \gamma\, v \\ \hline v^t \gamma & \bar{R}^t \bar{R} + v^t v \end{array}\right]$$

e portanto:

1) $p^t p = \gamma^2$. Lembrando que R^t é o fator de Cholesky associado a B, e é obtido através de : $QB^t = R$ temos de acordo com (1.6) que p é a r-ésima coluna de Q, se for considerado que A_r é a r-ésima coluna de B. Como Q é ortogonal temos que: $\gamma^2 = p^t p = 1$.

2) $p^t R = \gamma\, v = v$. De acordo com (1.6) $v = A_r^t$.

3) $R^t R = v^t v + \bar{R}^t \bar{R}$

$\bar{R}^t \bar{R} = R^t R - v^t v = BB^t - A_r A_r^t$ provando que $\bar{R}^t$ é o fator de Cholesky de $BB^t - A_r A_r^t$.

2ª etapa - Aqui, o fator de Cholesky obtido ao final da primeira etapa é modificado para $\hat{R}^t$, o fator de Cholesky associado a nova base $\hat{B}$. Construa a matriz $E_{(m+1)\times m}$ na forma:

$$E = \left[\begin{array}{c} \bar{R} \\ \hline A_s^t \end{array}\right]$$

Pré-multiplique E por uma sequência de matrizes de rotações de Givens P^j_{m+1}, j = 1..m. Isto anula os elementos da linha m+1 de E preservando a estrutura triângular superior de $\bar{R}$. A matriz E é transformada em:

$$\left[\begin{array}{c}\hat{R}\\ \hline 0 \ \ldots \ 0\end{array}\right]$$

e $\hat{R}^t$ é o fator de Cholesky associado a $\hat{B}$ pois:

$$\hat{Q} = \prod_{i=1}^{m} P^{i}_{m+1}$$

$$\hat{Q}\left[\begin{array}{c}\bar{R}\\ \hline A_s^t\end{array}\right] = \left[\begin{array}{c}\hat{R}\\ \hline 0 \ \ldots \ 0\end{array}\right]$$

$$\left(\hat{Q}\left[\begin{array}{c}\bar{R}\\ \hline A_s^t\end{array}\right]\right)^t \left(\hat{Q}\left[\begin{array}{c}\bar{R}\\ \hline A_s^t\end{array}\right]\right) = \left(\hat{Q}\left[\begin{array}{c}\bar{R}\\ \hline A_s^t\end{array}\right]\right)^t \left[\begin{array}{c}\hat{R}\\ \hline 0 \ \ldots \ 0\end{array}\right]$$

$$\left[\bar{R}^t \,|\, A_s\right] \left[\begin{array}{c}\bar{R}\\ \hline A_s^t\end{array}\right] = \left[\hat{R}^t \,|\, 0 \ldots 0\right] \left[\begin{array}{c}\hat{R}\\ \hline 0 \ \ldots \ 0\end{array}\right]$$

$$\bar{R}^t \bar{R} + A_s A_s^t = \hat{R}^t \hat{R}$$

$$\hat{R}^t \hat{R} = BB^t - A_r A_r^t + A_s A_s^t .$$

1.2.2 - MÉTODO SIMPLEX REVISADO COM GERAÇÃO DE COLUNAS [La]

Existem problemas de Programação Linear onde a matriz de restrições A possui o número de colunas muito superior ao número de linhas e não é possível resolvê-los pelo método Simplex usual

devido à impossibilidade de armazenar explicitamente a matriz. Se a matriz A possui uma estrutura tal que dado um vetor de preços π existe um algoritmo 'eficiente' para determinar ou uma coluna A_s satisfazendo:

$$\bar{c}_s = c_s - \pi A_s < 0$$

ou que não existe tal coluna, então é possível resolver o problema pelo método Simplex Revisado com geração de colunas. Em geral procura-se resolver, a cada iteração do Simplex, o problema:

$$\bar{c}_s = \min_j \{c_j - \pi A_j \text{ , onde } A_j \text{ é uma coluna de A}\}$$

se $\bar{c}_s \geq 0$ não existe candidata a entrar na base.

Este método consiste nas seguintes modificações do método Simplex Revisado (seção 1.2.1):

2) OTIMALIDADE - Para verificar se é possível melhorar a SBF atual, é ncessário encontrar uma coluna não básica que satisfaz :

$$\bar{c}_s = \min_j \{ c_j - \pi A_j \}$$

onde π é solução do sistema :

$$\pi B = c_B$$

Nos problemas tratados neste trabalho c_j é constante. Então tomar:

$$\bar{c}_s = c_s - \pi A_s = \min_j \{ c_j - \pi A_j \}$$

é equivalente a resolver o problema:

$$max \quad \pi A_j$$

$$s.a \quad A_j \text{ é uma coluna da matriz } A.$$

Se $\bar{c}_s < 0$ então A_s é candidata a entrar na base. Caso contrário, pare; a solução atual é ótima.

1.2.3 - MÉTODO SIMPLEX PARA VARIÁVEIS CANALIZADAS [KH]

Se o problema de Programação Linear a ser resolvido tem algumas variáveis limitadas superiormente é possível modificar o método Simplex para considerar estas restrições implicitamente diminuindo o número de linhas e colunas da base.

Seja o PL:

$$\begin{aligned} \min \quad & cx \\ s.a \quad & Ax = b \\ & x_j \geq 0 \quad j = 1..n \\ & x_j \leq u_j \quad j = 1..t,\ t \leq n \end{aligned}$$

Seja B uma submatriz não singular de A com m colunas. Supondo, sem perda de generalidade que é feita uma reordenação nas colunas de A convertendo-a em [B | L | U], então as soluções básicas associadas a B são expressas por : $x_B = \bar{b}$, $x_L = 0$, $x_U = u_U$ e $\pi = c_B B^{-1}$; onde $\bar{b} = B^{-1}b - B^{-1}Uu_U$.

As modificações necessárias no método simplex se resumem a:

2) OTIMALIDADE - Dada uma base B factível (isto é, $0 \leq x_B \leq u_B$) , a escolha da variável não básica candidata a entrar na base é feita considerando dois conjuntos:

$$\Psi_1 = \{ j : \bar{c}_j < 0 \quad e \quad j \in N \}$$

$$\Psi_2 = \{ j : \bar{c}_j > 0 \quad e \quad j \in U \}$$

se $\Psi_1 \cup \Psi_2 = \emptyset$ pare; a solução atual é ótima. Caso contrário selecione $s \in \Psi_1 \cup \Psi_2$ para entrar na base e faça :

$$\delta = \begin{cases} 1 & \text{se} \quad s \in \Psi_1 \\ -1 & \text{se} \quad s \in \Psi_2 \end{cases}$$

3) TESTE DA RAZÃO - Para Calcular o novo valor da variável candidata a entrar na base, escolha Δ como:

$$y = B^{-1} A_s$$

$$\Delta_1 = \min_i \{ \bar{b}_i / |y_{is}| , \qquad \delta y_{is} > 0 \}$$

$$\Delta_2 = \min_i \{ (u_{B(i)} - \bar{b}_i) / |y_{is}|, \quad \delta y_{is} < 0 \}$$

$$\Delta = \min_i \{ \Delta_1, \Delta_2, u_s \}$$

onde B(i) é o índice da i-ésima variável básica.

4) ATUALIZAÇÃO - A atualização é feita em duas etapas. Primeiro é atualizado o valor das variáveis:

$$x_s = x_s + \delta \Delta$$

$$\bar{b} = \bar{b} - \delta \Delta y$$

Se $\Delta = u_s$, então a base não muda, e inicia-se nova iteração. Em caso contrário, a decisão da variável que vai sair da base é feita escolhendo $r \in \Psi_3 \cup \Psi_4$, onde:

$$\Psi_3 = \{ i : y_{is} > 0 \quad e \quad \bar{b}_i = 0 \}$$

$$\Psi_4 = \{ i : y_{is} < 0 \ e \ \bar{b}_i = u_i \}$$

Então a base é atualizada retirando a r-ésima coluna básica $A_{B(r)}$ e colocando em seu lugar a coluna A_s.

1.2.4 - PROGRAMAÇÃO FRACIONÁRIA [La]

Para atender alguns requisitos da indústria, muitas vezes a formulação do problema de corte envolve a minimização da razão de duas funções lineares. Esta situação ocorre no problema do corte quando se deseja minimizar a perda relativa e há tolerância na demanda (cap. III). Nesta seção é mostrado que sob certas condições o método Simplex pode ser modificado para resolver o problema.

Seja o problema de Programação Fracionária:

$$\begin{aligned} \min \quad & f(x) = z^1(x)/z^2(x) \\ s.a \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \qquad (1.7)$$

onde $z^1(x) = c^1x + \alpha$ e $z^2(x) = c^2x + \beta$

Defindo a região factível como:

$$S = \{ x : Ax \leq b, \quad x \geq 0 \}$$

pode-se enunciar o seguinte teorema:

TEOREMA 1.2.1 - Suponha que $\mathbb{S}$ é limitado, e que $z^2(x) > 0, \forall\ x \in \mathbb{S}$, então f assume seu valor mínimo em um ponto extremo de $\mathbb{S}$.

Com este resultado, ver prova em Lasdon [La], é possível resolver o problema (1.7) percorrendo os pontos extremos do conjunto $\mathbb{S}$ de forma que ao passar de um ponto extremo para outro a função objetivo não aumente de valor (ou seja, usar a estratégia do método Simplex). Considere agora o seguinte teorema:

TEOREMA 1.2.2 - Seja $f(x) = z^1(x) / z^2(x)$, definida no conjunto $\mathbb{S}$. Sejam um ponto $x \in \mathbb{S}$ e v uma dada direção. Suponha que:

a) $z^2(x) > 0, \forall\ x \in \mathbb{S}$;

b) $\bar{x} + \alpha v \in \mathbb{S}$ para algum intervalo $0 < \alpha \leq \delta, \delta > 0$;

c) $g(0) = \left.\dfrac{df(\bar{x} + \alpha v)}{d\alpha}\right|_{\alpha=0} < 0$

Então $h(\alpha) = f(\bar{x} + \alpha v)$ é monótona decrescente para todo $0 < \alpha \leq \delta$.

Prova: Pelas suposições a) e b), $h(\alpha)$ é contínua e diferenciável no intervalo $[0, \delta]$. O teorema estará demonstrado se for provado que:

$$g(\alpha) = \frac{dh(\alpha)}{d\alpha} < 0 \quad \text{em} \quad [0, \delta].$$

Diferenciando $h(\alpha)$:

$$g(\alpha) = \frac{d}{d\alpha}\left(\frac{z^1(\bar{x} + \alpha v)}{z^2(\bar{x} + \alpha v)}\right)$$

$$= \frac{z^2(\bar{x} + \alpha v)\ z^1(v) - z^1(\bar{x} + \alpha v)\ z^2(v)}{[\ z^2(\bar{x} + \alpha v)\ ]^2}$$

$$= \frac{[\ z^2(\bar{x}) + \alpha z^2(v)\ ]\ z^1(v) - [\ z^1(\bar{x}) + \alpha z^1(v)\ ]\ z^2(v)}{[\ z^2(\bar{x} + \alpha v)\ ]^2}$$

como, $g(0) = \dfrac{z^2(\bar{x}) * z^1(v) - z^1(\bar{x}) * z^2(v)}{[\ z^2(\bar{x})\ ]^2}$

então: $g(\alpha) = \dfrac{[\ z^2(\bar{x})\ ]^2}{[\ z^2(\bar{x} + \alpha v)\ ]^2}\ g(0) < 0$, para $\alpha \in [0, \delta\ ]$.

Com este resultado, para encontrar um novo ponto extremo com valor da função objetivo menor, basta caminhar na direção da derivada direcional negativa. Se para algum ponto extremo todas as derivadas direcionais forem não negativas, então este ponto extremo é ótimo.

Para escolher o novo ponto extremo que melhora a função objetivo é necessário então calcular as derivadas direcionais $\bar{c}_j = \partial f / \partial x_j$ (f reescrita em função das variáveis não básicas), x_j variável não-básica. Se $\bar{c}_j < 0$ para alguma variável não-básica, então a função objetivo pode ser melhorada. Senão a solução atual é ótima.

Seja o conjunto de restrições:

$$Ax = b \Rightarrow [B\ N]\begin{bmatrix} x_B \\ x_N \end{bmatrix} = b \Rightarrow Bx_B + Nx_N = b$$

com as variáveis básicas escritas em função das não-básicas:

$$x_B = b - B^{-1}Nx_N \qquad (1.8)$$

Considere agora duas funções objetivos:

$$z^1(x) = c^1x = c_B^1 x_B + c_N^1 x_N$$

$$z^2(x) = c^2x = c_B^2 x_B + c_N^2 x_N$$

Reescrevendo-as, usando (1.8):

$$z^1(x) = c_B^1b - c_B^1 B^{-1}Nx_N + c_N^1 x_N = c_B^1b + (c_N^1 - c_B^1 B^{-1}N)x_N$$

$$z^2(x) = c_B^2b - c_B^2 B^{-1}Nx_N + c_N^2 x_N = c_B^2b + (c_N^2 - c_B^2 B^{-1}N)x_N$$

fazendo :

$$\bar{z}^1 = c_B^1b\ , \quad \bar{c}_j^1 = c_j^1 - \pi^1A_j\ , \quad \pi^1 = c_B^1B^{-1} \qquad (1.9)$$

$$\bar{z}^2 = c_B^2b, \quad \bar{c}_j^2 = c_j^2 - \pi^2A_j\ , \quad \pi^2 = c_B^2B^{-1} \qquad (1.10)$$

tem-se:

$$z^1(x) = \bar{z}^1 + \bar{c}_N^1 x_N$$

$$z^2(x) = \bar{z}^2 + \bar{c}_N^2 x_N$$

Calculando então, as derivadas direcionais:

$$\bar{c}_j = \frac{\partial \left(z^1(x) / z^2(x) \right)}{\partial x_j}$$

$$= \frac{\bar{z}^2 \; \bar{c}^1_j - \bar{z}^1 \; \bar{c}^2_j}{[\; \bar{z}^2 \;]^2} \qquad (1.11)$$

Assim, para resolver um problema de Programação Fracionária que satisfaça as condições do TEOREMA 1.2.1, aplica-se o algoritmo Simplex Revisado (seção 1.2.1) considerando o cálculo dos custos relativos das variáveis não-básicas feitos através da equação (1.11). Supondo que não há degeneração, a função objetivo decresce a cada iteração, nenhuma base é repetida, e uma solução ótima é obtida em um número finito de iterações.

CAPÍTULO II

2 - O PROBLEMA DA MOCHILA

Considere uma mochila com capacidade limitada e diversos itens com pesos e valores conhecidos. O problema da mochila consiste em determinar um subconjunto desses itens cujo peso total não exceda a capacidade da mochila e cujo valor total seja o maior possível.

Apesar de apresentar a mais simples estrutura para um programa inteiro, o problema da mochila tem sido estudado intensivamente porque:

a) modela muitos problemas práticos tais como: seleção de projetos ou investimentos de capital, problemas de alocação/orçamento ;

b) aparece como um subproblema na modelagem de vários problemas com estrutura mais complexa tais como: o corte unidimensional e o corte bidimensional, tratados nos próximos capítulos.

Os métodos mais usados para resolver o PROBLEMA DA MOCHILA são :

a) Enumeração implicita - O de uso mais frequente é o método "Backtraking" (os subproblemas são resolvidos à medida que vão sendo gerados). Também é utilizado o método Particionar e Limitar ("branch and bound" - os subproblemas gerados são armazenados em uma lista para serem resolvidos posteriormente). Os limitantes são obtidos através da relaxação do problema em programação linear, da

relaxação lagrangeana, ou da relaxação por substituição.

b) Programação Dinâmica.

c) Heurísticas - São usadas ou quando não existe um algoritmo exato ou quando existe apenas algoritmos não eficientes.

Serão apresentados neste capítulo as seguintes variações deste problema:

2.1 - Mochila 0/1 (0/1 Kanapsack)

2.2 - Mochila Inteiro (Integer Kanapsack)

2.3 - Múltiplas Mochilas (Multiple Kanapsacks)

2.4 - Número Mínimo de Mochilas (Bin Packing).

Nos capítulos III e IV será discutido o problema da Mochila Multidimensional (Multidimensional Kanapsack) onde a cada item é associado um valor e diversas dimensões (largura, e largura/comprimento).

2.1 - O PROBLEMA DA MOCHILA 0/1 [MT]

Suponha que o conjunto de n itens é tal que não existem dois itens com o mesmo par de valor e peso. Isto implica que existem apenas duas possibilidades para cada item: ser colocado ou não dentro da mochila. Matemáticamente, o PROBLEMA DA MOCHILA 0/1 (PM) pode ser formulado através do seguinte programa linear inteiro:

$$\max \quad \sum_i v_i \; y_i$$

$$s.a \quad \sum_i p_i \; y_i \leq C \qquad (2.1)$$

$$y_i \in \mathbb{B} \qquad (2.2)$$

onde:

v_i = valor do item i (inteiro positivo)

p_i = peso do item i (inteiro positivo)

C = capacidade da mochila

$\mathbb{B}$ = { 0, 1 }

y_i = 0 (1) indica que o item i foi excluído (incluído) na mochila.

De maneira compacta podemos escrever:

$$PM : \max \{ vy : py \leq C, \; y \in \mathbb{B}^n \}$$

Como o peso e o valor de cada item são positivos considere sem perda de generalidade que:

$$\max \{ p_i \} < C \quad e \quad \sum_i p_i > C$$

São apresentados a seguir limitantes para o problema, procedimentos de redução do número de variáveis, e dois métodos de resolução para o problema.

2.1.1 - LIMITANTES

Os limitantes superiores para o problema desenvolvidos nesta seção são baseados na relaxação do problema em programa linear e na relaxação lagrangeana.

Suponha que os itens estão ordenados em ordem não decrescente de valor por unidade de peso, ou seja:

$$v_i / p_i \geq v_{i+1} / p_{i+1} \qquad (2.3)$$

e defina por item crítico o item s que satisfaz:

$$s = \min \{ k : \sum_{i=1}^{k} p_i > C \}$$

A relaxação do Problema da mochila 0/1 em um programa linear é:

$$PM : \max \{ vy : py \leq C, \quad 0 \leq y \leq 1, \}$$

também chamado o PROBLEMA DA MOCHILA CONTÍNUO (PM_L):

As condições de folgas complementares (seção 1.1), a restrição (2.2) e a ordenação dos itens de acordo com (2.3) permitem escrever uma solução ótima para PM_L da forma:

$$\bar{y}_i = 1 \qquad i = 1..s-1$$

$$\bar{y}_i = 0 \qquad i = s+1..n$$

$$\bar{y}_s = \bar{C}/p_s \qquad \text{onde } \bar{C} = (C - \sum_{i=1}^{s-1} p_i)$$

A partir desta solução, diversos limitantes superiores para PM podem ser deduzidos. O mais simples, obtido diretamente da solução ótima do problema contínuo é:

$$u_1 = \lfloor z(PM_L) \rfloor = \sum_{i=1}^{s-1} v_i + \lfloor \bar{C} * v_s / p_s \rfloor \qquad (2.4)$$

onde z(P) denota o valor ótimo de um problema P e $\lfloor r \rfloor$ é o piso de r; isto é, o maior inteiro menor ou igual a r.

Um limitante superior melhor pode ser obtido considerando-se ou não a inserção do item s na mochila. Se o item crítico não é inserido, o valor da solução não excede a:

$$b_1 = \sum_{i=1}^{s-1} v_i + \lfloor \bar{C} * v_{s+1} / p_{s+1} \rfloor$$

que corresponde à inserção do elemento com maior razão v_i/p_i. Se o item crítico é enserido , pelo menos um dos primeiros s-1 itens é removido, assim a melhor solução será dada por:

$$b_2 = \sum_{i=1}^{s-1} v_i + v_s - \lfloor (p_s - \bar{C}) * v_{s-1}/p_{s-1} \rfloor$$

onde se supõe que o elemento retirado possui exatamente o valor necessário de p_i (isto é $p_s - \bar{C}$) e o pior valor possível de v_i/p_i (isto é v_{s-1}/p_{s-1}). O limitante é então:

$$u_2 = \max (b_1, b_2) \qquad (2.5)$$

As considerações feitas para se chegar a u_2 podem ser exploradas para se encontrar limitantes mais restritivos que u_1 e u_2. Resolvendo o PROBLEMA DA MOCHILA CONTÍNUO com a restrição adicional de $y_s = 0$ ou $y_s = 1$ obtém-se:

$$b_3 = \lfloor z(PM_L \text{ com } y_s = 0) \rfloor$$

$$b_4 = \lfloor z(PM_L \text{ com } y_s = 1) \rfloor$$

o limitante é então :

$$u_3 = \max (b_3, b_4).$$

Outros limitantes superiores podem ser determinados por meio da relaxação lagrangeana. Dado $\lambda > 0$, a relaxação lagrangeana de PM é definida por:

$$PM_\lambda: \max \{ vy + \lambda(C - py) : y \in \mathbb{B}^n \}$$

ou:

$$PM_\lambda: \max \{ \lambda C + \tilde{v}y) : y \in \mathbb{B}^n \}$$

onde $\tilde{v}_i = v_i - \lambda p_i$. Uma solução ótima $y^* = (y_i^*)$ de PM_λ é:

$$y_i^* = \begin{cases} 1 & \tilde{v}_i > 0 \\ 0 & \tilde{v}_i < 0 \\ 0 \text{ ou } 1 & \tilde{v}_i = 0. \end{cases}$$

Observe que este problema possui a propriedade de integralidade, ou seja, o valor de $z(PM_\lambda)$ não se altera quando há relaxação da restrição $y_i = 0/1$ em $0 \leq y_i \leq 1$:

$$z(PM_{\lambda L}) = z(PM_\lambda) \qquad \lambda > 0.$$

O valor mínimo de $z(PM_\lambda)$ ocorre quando $\lambda = \tilde{\lambda} = p_s/w_s$, pois:

$$\tilde{v}_i > 0 \quad i = 1..s-1 \text{ e}$$
$$\tilde{v}_i = 0 \quad i = s$$
$$\tilde{v}_i < 0 \quad i = s+1..n$$

e então pode-se afirmar que:

$$z(PM_\lambda) \geq z(PM_{\tilde{\lambda}}) = z(PM_{L\tilde{\lambda}}) = z(PM_L) \geq z(PM)$$

e portanto:

$$\tilde{y}_i = \bar{y}_i \qquad i = 1..s-1,s+1..n$$

onde $\tilde{y}$ é a solução ótima de $z(PM_{\tilde{\lambda}})$ e $\bar{y}$ é a solução ótima de $z(PM_L)$.

Para obter uma solução inteira para PM a partir da solução contínua, $\bar{y}$, é necessário fazer ou $\bar{y}_s = 0$ ou então $\bar{y}_s = 1$ e neste último caso pelo menos uma das outras variáveis, digamos y_i, deve tomar o valor $1-\bar{y}_i$. Se a última alternativa for executada, para $i \neq s$, temos:

$$z(PM_L \text{ com } y_i = 1-\bar{y}_i) \leq z(PM_\lambda \text{ com } y_i = 1-\tilde{y}_i)$$

mas, $\tilde{v}_i$ representa um decrescimento de $z(PM_\lambda)$ correspondente à mudança do valor de $y_i = \tilde{y}_i$ para $y_i = 1 - \tilde{y}_i$, então podemos escrever:

$$\begin{aligned} z(PM \text{ com } y_i = 1-\tilde{y}_i) &\leq z(PM_{\tilde{\lambda}} \text{ com } y_i = 1-\tilde{y}_i) \\ &= z(PM_{\tilde{\lambda}}) - \lfloor \tilde{v}_i \rfloor = z(PM_L) - \lfloor \tilde{v}_i \rfloor . \end{aligned}$$

Assim um limitante superior é:

$$u_4 = \max \langle z(PM_L) - \bar{C}\, v_s/p_s),\ \lfloor z(PM_L) - \min \langle \tilde{v}_i,\ i \neq s,\ i = 1..n \rangle \rfloor \rangle$$

Através das definições dos limitantes feitas acima pode se provar as seguintes relações :

1) $u_2 \leq u_1$

2) $u_3 \leq u_2$ pois : $b_3 \leq b_1$ e $b_4 \leq b_2$

3) $u_4 \leq u_1$

Embora todos estes limitantes tenham complexidade computacional O(n), se os itens estiverem ordenados seus desempenhos médios podem ser bastante diferentes.

Na implementação de um algoritmo de enumeração implícita, o limitante a ser calculado em cada nó deve ser escolhido considerando-se que u_1, u_2 e u_3 podem ser facilmente calculados pois o valor de s geralmente muda muito pouco de um nó para outro, enquanto que u_4 necessita da avaliação do valor de $\tilde{v}_i$ para todas as variáveis correntes y_i não fixadas.

2.1.2 - PROCEDIMENTOS DE REDUÇÃO

O número de variáveis binárias de um determinado problema pode ser reduzido através de técnicas que fixam o valor ótimo do maior número possível de variáveis. O procedimento de redução visto aqui, particiona o conjunto N = { 1..n } em três subconjuntos:

$I0 = \{ i \in N : y_i = 0$ em cada solução ótima para PM $\}$

$I1 = \{ i \in N : y_i = 1$ em cada solução ótima para PM $\}$

$I = N - I0 \cup I1$.

O problema original pode então ser transformado na forma reduzida:

$$PM_R: \max \{ vy + \hat{v} : py \leq \hat{C},\ y_i \in \mathbb{B},\ i \in I \}$$

onde: $\hat{v} = \sum_{i \in I1} v_i$ e $\hat{C} = C - \sum_{i \in I1} p_i$

Os subconjuntos I1 e I0 são construídos, considerando-se as implicações de se fazer uma variável receber valor 0 ou 1. Se a atribuição de um valor a = 0/1 para y_i implica em um subproblema infactível ou em uma solução ótima com valor inferior ao de uma solução já conhecida, pode se concuir y_i = 1-a, desde que esta é a única escolha que pode conduzir a uma solução factível ou melhor. Para o problema PM, os mais eficientes procedimentos de redução são obtidos avaliando estas implicações através do cálculo de limitantes superiores.

Seja $\hat{y}$ uma solução factível para o problema PM e $\hat{z}$ seu correspondente valor ($\hat{z}$ representa um limitante inferior para o valor ótimo z^*). Além disso, para $i \in N$, seja z_i (resp. $\bar{z}_i$) o limitante superior para PM com a restrição adicional de $y_i = 1$ (resp. $\bar{y}_i = 0$). Então:

$$I0 = \{ i \in N : z_i \leq \hat{z} \}$$
$$I1 = \{ i \in N : \bar{z}_i \leq \hat{z} \}.$$

A eficiência dos procedimentos de redução vai depender das técnicas usadas no cálculo de $\hat{z}$, z_i e $\bar{z}_i$. Considerando que o cálculo dos limitantes z_i e $\bar{z}_i$ pode ser feito através de resolução de PM_L com $y_i = 1$ e PM_L com $y_i = 0$ e que isto requer tempo computacional O(n) para encontrar o item crítico s_i e $\bar{s}_i$ a complexidade global do procedimento de redução pode ser fixada em $O(n^2)$.

Um bom decréscimo no tempo médio do procedimento pode ser obtido observando que é inútil calcular z_i (resp. $\bar{z}_i$) se $y_i = 1$ (resp. $\bar{y}_i = 0$) na solução correspondente ao limitante superior do problema original PM.

A cardinalidade do subconjunto I pode ser decrescida considerando:

$$I0 = I0 \cup \{ i \in I : p_i > \bar{C} \}$$

e, se:

$$\sum_{i \in I} p_i \leq \bar{C} ,$$

então:

$$I1 = I1 \cup I.$$

Este procedimento de redução em I deve ser repetido até que nenhuma variável mais possa ser fixada.

2.1.3 - METODOS DE ENUMERAÇÃO IMPLICITA

Muitos algoritmos de enumeração implícita têm sido propostos nas últimas décadas para encontrar a solução exata do PROBLEMA DA MOCHILA 0/1. Estes algoritmos diferem nas técnicas de limitação, na estratégia de partição, na aplicação dos critérios de infactibilidade e domínio e no cálculo dos limitantes superiores.

Os métodos mais eficientes usam um esquema de partição binário, onde em cada nó é selecionado um item ainda não fixado, gerando dois nós descendentes com a fixação de y_i em 1 ou 0. A busca continua no nó associado com a inserção do i-ésimo item na solução ($y_i = 1$), isto é, a partir do nó com maior limitante superior. Em seguida serão mostrados dois dos mais eficientes algoritmos propostos por Horowitz e Sahni [HS] e por Martello e Toth [MT1].

O algoritmo proposto em Horowitz e Sahni [HS] começa pela ordenação dos itens em ordem não decrescente de valor por unidade de peso. Toma como primeira solução corrente a solução do PROBLEMA DA MOCHILA CONTÍNUO com $y_s = 0$. O movimento progressivo ("forward") consiste na colocação do maior número possível de itens na solução corrente observando a restrição (2.1). O movimento regressivo ("backtracking") consiste em remover o último item colocado na solução corrente, ou seja, em retirar o item com menor valor v_i/p_i. Enquanto o movimento progressivo é realizado, o limitante superior u_1 (eq. 2.4) correspondente à solução corrente é calculado e comparado com a melhor solução até o momento de forma a verificar se um movimento progressivo pode levar a uma solução melhor. Se isto ocorre, então um novo movimento progressivo é realizado, senão, é feito um movimento regressivo. Quando o último item for considerado, a solução está completa e pode atualizar o valor da melhor solução até o momento. O algoritmo para quando não é possível realizar um movimento regressivo.

Em Martello e Toth [MT1] o algoritmo proposto difere do anterior nos seguintes aspectos:

i) o limitante superior u_2 (eq. 2.5) é usado no lugar de u_1.

ii) o movimento progressivo associado com a inserção do i-ésimo item é dividido em duas fases: construir uma nova solução e salvar a solução corrente. Na primeira fase é definido o maior número N_i de elementos consecutivos que podem ser inseridos na solução corrente a partir do i-ésimo item e é calculado o limitante superior u_i correspondente a inserção do i-ésimo item. Se u_i é menor ou igual ao valor da melhor solução até o momento, o movimento regressivo ocorre imediatamente. Se u_i for maior a

segunda fase, ou seja, a inserção dos itens do conjunto N_i na solução corrente, é feita apenas se o valor desta nova solução não representar o máximo que pode ser obtido com a inserção do i-ésimo item. Caso contrário, a melhor solução até o momento é atualizada mas a solução corrente não, de forma que movimentos regressivos nos itens do conjunto N_i sejam evitados.

iii) Um procedimento particular de movimento progressivo, é realizado sempre que antes do movimento regressivo no i-ésimo item a capacidade residual da mochila, $\bar{C}$, não permite a inserção de nenhum item posterior ao i-ésimo. O procedimento é baseado na seguinte consideração: a solução corrente pode ser melhorada apenas se o i-ésimo item for substituído por outro item com valor maior e peso menor o suficiente para permitir sua inserção, ou por pelo menos dois itens possuindo peso total menor ou igual a $p_i + \bar{C}$. Desta forma, geralmente é possível eliminar a maior parte dos nós inúteis gerados nos níveis menores da árvore de decisão.

iv) Os limitantes superiores associados aos nós da árvore de decisão são calculados baseado no armazenamento da informação relacionada com a solução corrente. Suponha que a solução corrente foi construída pela inserção do i-ésimo ao r-ésimo item, então, quando na tentativa de construir uma nova solução a partir do j-ésimo item ($i < j < r$) nenhuma inserção ou remoção ocorre para os itens que precedem o j-ésimo, é possível inserir pelo menos os itens a partir do j-ésimo até o r-ésimo na solução corrente.

2.1.4 - MÉTODOS DE PROGRAMAÇÃO DINÂMICA

Programação Dinâmica é um método que pode ser usado quando a solução de um problema pode ser obtida como o resultado de uma sequência de decisões.

No caso do PROBLEMA DA MOCHILA 0/1 a k-ésima decisão a ser tomada consiste em atribuir valor 0 ou 1 para a variável y_k, k = 1..n.

Considere $z_k(c)$ como o valor máximo da função objetivo usando apenas as primeiras k (k = 1..n) variáveis e com limitação de capacidade c (c = 1..C). Isto é:

$$PM_k^c = \max \{ vy : py \leq c , y \in \mathbb{B}^k \}$$

$$z_k(c) = PM_k^c$$

$$y_k(c) = y_k \qquad \text{onde, } y_1..y_k \text{ é solução ótima de } PM_k^c$$

Da definição acima segue que :

$$PM_1^c = \max \{ v_1 y_1 : p_1 y_1 \leq c, y_1 \in \mathbb{B} \},$$

isto é:

$$z_1(c) = \begin{cases} 0 & , c = 0..p_1-1 \\ v_1 & , c = p_1..C \end{cases}$$

$$y_1(c) = \begin{cases} 0 & , c = 0..p_1-1 \\ 1 & , c = p_1..C \end{cases}$$

Define-se então para o k-ésimo estágio (k = 2..n) e para c = 0..C as seguintes equações recursivas:

$$z_k(c) = \begin{cases} z_{k-1}(c) & c = 0..p_k-1 \\ \max\{z_{k-1}(c),\ z_{k-1}(c-p_k) + v_k\} & c = p_k..C \end{cases}$$

$$y_k(c) = \begin{cases} 0 & \text{se } z_k(c) = z_{k-1}(c) \\ 1 & \text{se } z_k(c) > z_{k-1}(c) \end{cases}$$

a solução ótima será dada por:

$$z^* = z_n(C)$$

com: $$y_k^* = y_k(c_k) \quad \text{onde } c_{k-1} = c_k - y_k^* p_k$$

Observe que cada estágio k (k= 1..n) pode ser identificado por uma sequência, S^k, de pares ordenados do tipo (c, $z_k(c)$) em ordem crescente do valor da primeira componente do par. Caso hajam dois pares ($\tilde{c}$, $z_k(\tilde{c})$) e ($\bar{c}$, $z_k(\bar{c})$) tais que $\tilde{c} < \bar{c}$ e $z_k(\tilde{c}) \geq z_k(\bar{c})$, então diz-se que o par ($\tilde{c}$, $z_k(\tilde{c})$) domina o par ($\bar{c}$, $z_k(\bar{c})$) e este último deve ser eliminado, pois é desnecessário. Cada estágio é então definido considerando apenas os pares não dominados e viáveis (isto é $0 \leq c \leq C$). Desta forma, a sequência S^k define uma função "escada" não-decrescente.

Para encontrar a solução ótima, Horowitz e Sahni [HS1] apresentam o seguinte algoritmo.

Inicialmente é construída a sequência $S^1 = \{ (0, 0), (p_1, v_1) \}$ que é a solução da equação $z_1(c)$. Em seguida é construída a sequência S^2 a partir de S^1 e assim, sucessivamente, até considerar todos os itens. O valor ótimo é tomado como sendo o último elemento da sequência S^n.

Dada a sequência de pares do estágio S^{k-1}, contrói-se a sequência de pares não dominados e viáveis S^k, através da união adequada ("merge") de S^{k-1} com a sequência obtida somando o par (p_k, v_k) aos pares de S^{k-1} .

Toth [T] propôs um algoritmo onde limitantes superiores são usados para eliminar os estados que não levam a soluções ótimas e os estados inutilizados, ou seja, os estados que não serão utilizados nos estágios seguintes. O método combina ainda as equações recursivas progressiva e regressiva considerando o número de estados não dominados e o valor de C.

2.2 - PROBLEMA DA MOCHILA INTEIRO

O problema da mochila inteiro é uma extensão do problema apresentado na seção 2.1 onde se considera que estão disponíveis vários itens com o mesmo par de valor e peso. Desta forma:

$$PMI : \max \{ vy : py \leq C, y \in \mathbb{Z}_+^n \}$$

Para encontrar a solução deste problema serão apresentados nas seções seguintes dois métodos: Programação Dinâmica e Enumeração Implicita.

2.2.1 - LIMITANTES

Um limitante superior para este problema pode ser obtido através da solução do problema PMI relaxado em programação linear (PMI_L).

$$PMI_L: \max \{ vy : py \leq C, y \in \mathbb{R}^n_+ \}$$

Considerando que os itens estão ordenados em ordem não decrescente de valor por unidade de peso, ou seja de acordo com eq. (2.3) a solução para PMI_L é dada por:

$$y_1 = C / p_1 \qquad e \qquad y_k = 0, \quad k = 2..n.$$

Um limitante superior é dado por:

$$u = \lfloor v_1 (C/p_1) \rfloor.$$

2.2.2. - METODOS DE ENUMERAÇÃO IMPLICITA

O algoritmo abaixo foi elaborado com base no algoritmo proposto em Horowitz e Sahni [HS] descrito na seção 2.1.3.

O movimento progressivo para o presente problema vai consistir em inserir o maior número possível de cada elemento na solução corrente. O movimento regressivo será retirar uma vez o último item inserido na solução corrente.

Antes de inserir um elemento na mochila, é verificado se esta inserção vai melhorar o valor da solução corrente. Se não melhorar, o movimento regressivo ocorre imediatamente.

Sempre que houver uma solução completa, antes de efetuar o movimento regressivo, é verificado se ela pode atualizar o valor da melhor solução até o momento.

Quando nenhum movimento regressivo adicional for possível o algoritmo termina.

2.2.3 - MÉTODOS DE PROGRAMAÇÃO DINÂMICA

O algoritmo aqui apresentado foi desenvolvido com base no algoritmo para a mochila 0/1, descrito na seção 2.1.4.

Considere como $z_k(c)$ o valor máximo da função objetivo usando apenas as primeiras k (k = 1..n) variáveis e com limitação de capacidade c (c = 1..C). Isto é:

$$PMI_k^c = \max \{ vy : py \leq c,\ y \in \mathbb{Z}_+^k \}$$

$$z_k(c) = z(PMI_k^c)$$

e

$$y_k(c) = y_k \quad \text{onde,} \quad y_1..y_k \quad \text{é solução ótima de } PMI_k^c$$

Da definição acima segue que :

$$PMI_1^c = \max \{ v_1y_1 : p_1y_1 \leq c,\ y_1 \in \mathbb{Z}_+ \}$$

isto é:

$$z_1(c) = \{ v_1(\lfloor c/p_1 \rfloor) \}, \qquad c = 0..C$$

$$y_1(c) = \{ \lfloor c/p_1 \rfloor \}, \qquad c = 0..C$$

Então para k = 2..n e para c = 0..C podemos definir as seguintes equações recursivas:

$$z_k(c) = \max \{ jv_k + z_{k-1}(c - jp_k) , \quad j = 0,1..\lfloor c/p_k \rfloor \}$$

$$y_k(c) = j, \quad \text{onde } z_k(c) = \{ jv_k + z_{k-1}(c - jp_k)$$

a solução ótima é encontrada recursivamente através de:

$$z^* = z_n(C) \qquad\qquad c_n = C$$

$$y_k^*(C) = y_k(c_k), \quad c_{k-1} = c_k - y_k^* p_k$$

Cada estágio k = 1..n pode ser identificado por uma sequência, S^k, de ternos ordenados do tipo $(c, z_k(c), y_k(c))$ em ordem crescente pelo valor da primeira componente do terno. Caso hajam dois ternos $(\tilde{c}, z_k(\tilde{c}), y_k(\tilde{c}))$ e $(\bar{c}, z_k(\bar{c}), y_k(\bar{c}))$ tais que $\tilde{c} < \bar{c}$ e $z_k(\tilde{c}) \geq z_k(\bar{c})$, então diz-se que o terno $(\tilde{c}, z_k(\tilde{c}), y_k(\tilde{c}))$ domina o terno $(\bar{c}, z_k(\bar{c}), y_k(\bar{c}))$ e este último deve ser eliminado, pois é desnecessário. Cada estágio é então definido por uma sequência de ternos dominantes e viáveis (isto é $0 \leq c \leq C$).

O algoritmo constrói inicialmente a sequência $S^1 = \{ (0, 0, 0), (p_1, v_1, 1)..(jp_1, jv_1, j) \}$ onde $j = 2..\lfloor C/p_1 \rfloor$, solução de PMI_1^C. Em seguida é construída a sequência S^2 a partir de S^1 e procede-se assim, sucessivamente, até considerar todos os itens. O valor ótimo $z_n(C)$ é tomado como sendo o último elemento da sequência S^n.

Dada a sequência de ternos do estágio S^{k-1}, contrói-se as sequências de ternos não dominados e viáveis S_j^k, $j = 0..\lfloor C/p_k \rfloor$, somando o terno $\{ jp_k, jv_k, j)$ aos ternos de S^{k-1} respectivamente.

S^k é obtido unindo as sequências S^k_j observando a ordenação dos ternos pela primeira componente.

Na implementação realizada as seqências de ternos S^k_j não são obtidas explicitamente, apenas as sequências S^k são armazenadas. S^k é obtido através da união apropriada ("merge") de S^{k-1} com a sequência obtida pela adição do terno $(p_k, v_k, 1)$ aos ternos de S^k. Neste caso S^k é inicializado com o terno (0, 0, 0). Deve-se ressaltar que na implementação do algoritmo de Horowitz e Sahni [HS1] para a machila 0/1 (seção 2.1.4) o conjunto S^k é obtido pela união apropriada de S^{k-1} com $S^{k-1} \oplus (p_k, v_k)$.

2.3 - PROBLEMA DAS MÚLTIPLAS MOCHILAS [MT]

O Problema das Múltiplas Mochilas 0/1 (PMM) é uma generalização do problema apresentado na seção 2.1 onde estão diponíveis m mochilas com capacidades $C_1, C_2, \ldots, C_m$.

Este problema pode representar várias situações na indústria tais como: carregar m navios com n "containers", encher m tanques com n líquidos que não podem ser misturados, ou ainda cortar m itens unidimensionais em n pedaços de tamanhos pré-fixados menores.

O problema consiste em determinar y_{ji} (se o item i deve ser incluído ou não na mochila j) de forma a maximizar o valor total dos itens incluídos:

$$\text{PMM} : \max \quad \sum_{j=1}^{m} \sum_{i=1}^{n} v_i \; y_{ji}$$

s. a

$$\sum_{i=1}^{n} p_i \, y_{ji} \leq C_j \quad j = 1..m \qquad (2.9)$$

$$\sum_{j=1}^{m} y_{ji} \leq 1 \quad i = 1..n \qquad (2.10)$$

$$y_{ji} = 0/1 \quad i = 1..n \quad e \quad j = 1..m$$

Fazendo m = 1 tem-se o problema PM.

Neste problema a restrição (2.10) implica que cada item pode ser alocado em apenas uma das mochilas.

É comum fazer as seguintes suposições:

$$\min_i \{ p_i \} < \min_j \{ C_j \}$$

$$\max_i \{ p_i \} < \max_j \{ C_j \}$$

$$\sum_{i=1}^{n} p_i > \max_j \{ C_j \}$$

E considerar também que os itens estão ordenados por valor específico (eq. 2.3) e as mochilas em ordem não crescente de capacidade:

$$C_j \leq C_{j+1}.$$

Os algoritmos para o problema PMM podem ser divididos em duas classes dependendo dos valores de n e m. O problema é de fato difícil no sentido de que todos os algoritmos conhecidos apresentam tempo computacional não polinomial com respeito a m e n fazendo com que os algoritmos sejam orientados para o caso de baixos valores da razão n/m ou para o caso de altos valores desta razão.

São apresentados a seguir limitantes superiores para o problema, procedimentos para reduzir o número de variáveis e alguns algoritmos para obter a solução exata e a solução aproximada para o problema considerando o caso de altos valores da razão n/m.

2.3.1 - LIMITANTES

Dois métodos de relaxação são geralmente usados para o cálculo de limitantes para o problema PMM: a relaxação lagrangeana e a relaxação por substituição ("Surrogate relaxation").

A relaxação lagrangeana é obtida através de uma penalização na função objetivo utilizando a restrição de que cada item é colocado em apenas uma mochila - (2.10). Assim, a relaxação lagrangeana relativa a um vetor não negativo λ é dada por:

$$PMM_{\lambda} : \max \quad \sum_{j=1}^{m} \sum_{i=1}^{n} v_i \, y_{ji} - \sum_{i=1}^{n} \lambda_i \left(\sum_{j=1}^{m} y_{ji} - 1 \right) \qquad (2.11)$$

$$\text{s. a} \quad \sum_{i=1}^{n} p_i \, y_{ji} \leq C_i \qquad j = 1..m$$

$$y_{ji} = 0/1 \qquad i = 1..n$$

$$j = 1..m.$$

juntando os termos comuns, a função objetivo (2.11) pode ser reescrita como:

$$\max \quad \sum_{i=1}^{n} \lambda_i + \sum_{j=1}^{m} \sum_{i=1}^{n} (v_i - \lambda_i) y_{ji}.$$

Encontrando um valor para o vetor λ, o problema PMM relaxado pode ser decomposto em uma coleção de m problemas da Mochila:

$$z_j = \max \sum_{i=1}^{n} (v_i - \lambda_i) y_{ji}$$

$$s.a \quad \sum_{i=1}^{n} p_i y_{ji} \leq C_j$$

$$y_{ji} = 0 \ / 1$$

para $j = 1..m$

e o valor ótimo para PMM_λ será dado por:

$$\sum_{i=1}^{n} \lambda_i + \sum_{j=1}^{m} z_j .$$

Diferente da relaxação lagrangeana que faz com que a função objetivo absorva um conjunto de restrições, a estratégia da relaxação por substituição consiste em, como o próprio nome indica, substituir a conjunto original de restrições por uma outra restrição chamada de restrição substituta.

Então, dado um vetor não negativo π, a relaxação por substituição para o problema PMM relativa a este vetor é dada por:

$$PMM_\pi : \max \sum_{j=1}^{m} \sum_{i=1}^{n} v_i y_{ji}$$

$$s.a \quad \sum_{j=1}^{m} \pi_j \sum_{i=1}^{n} p_i y_{ji} \leq \sum_{j=1}^{m} \pi_j C_i$$

$$\sum_{j=1}^{m} y_{ji} \leq 1 \qquad i = 1..n$$

$$y_{ji} \in \mathbb{B}$$

Para obter limitantes bons, os valores dos vetores λ e π devem ser tomados de forma a minimizar o valor de PMM_{λ} e PMM_{π}. Este valor para λ pode ser encontrado através do uso de subgradientes; e o valor ótimo de π é provado por Martello e Toth [MT2] ser $\pi_i = k$ ($k \in \mathbb{Z}^+$) para i = 1..m.

Considere o problema PMM relaxado em programação linear:

$$PMM_L : \max \sum_{j=1}^{m} \sum_{i=1}^{n} v_i \, y_{ji}$$

s. a

$$\sum_{i=1}^{n} p_i \, y_{ji} \leq C_j \qquad j = 1..m$$

$$\sum_{j=1}^{m} y_{ji} \leq 1 \qquad i = 1..n$$

$$0 \leq y_{ji} \leq 1 \qquad i = 1..n$$

$$\text{e } j = 1..m$$

cuja solução ótima primal é:

$$\sum_{j=1}^{m} \bar{y}_{ji} = \begin{cases} 1 & i < t \\ \left(\sum_{j=1}^{m} C_j - \sum_{i=1}^{t-1} p_i \right) / p_t & i = t \\ 0 & i > t \end{cases}$$

onde t é o menor índice que satisfaz:

$$\sum_{i=1}^{t} p_i > \sum_{j=1}^{m} C_j .$$

e a solução dual associada:

$$\bar{\lambda}_i = \begin{cases} v_i - p_i (v_t / p_t) & i < t \\ 0 & i \geq t \end{cases}$$

e:

$$\bar{\pi}_j = v_t / p_t \qquad j = 1..m \; .$$

Como os valores das variáveis duais podem ser facilmente obtidos é conveniente o seu uso para obter limitantes superiores para PMM através de PMM_{λ} e PMM_{π} . Assim, $\lambda_i = \bar{\lambda}_i$ i = 1..n e $\pi_j = \bar{\pi}_j$ j = 1..m .

Como foi visto, a relaxação lagrangeana decompõe o problema PMM em uma série de problemas da mochila 0/1. De forma similar, fazendo uma mudança de variáveis do tipo $x_i = \sum_{j=1}^{m} y_{ji}$ (i = 1..n), e atribuindo um valor constante para as componentes de π, a relaxação de substituição pode ser expressa como um único problema da mochila 0/1:

$$\max \quad z = \sum_{i=1}^{n} v_i \, x_i$$

s.a

$$\sum_{i=1}^{n} p_i \, x_i \leq \sum_{j=1}^{m} C_j$$

$$x_i = 0/1 \quad i = 1..n$$

Nenhuma das duas relaxações domina a outra. Em geral, espera-se que PMM_{π} forneça um limitante melhor quando m é pequeno ou a razão n/m é grande. Apesar de outros multiplicadores fornecerem limitantes melhores que $\bar{\lambda}$ e $\bar{\pi}$, o fato das variáveis duais serem facilmente calculadas faz com que o esforço computacional requerido para encontrar outros multiplicadores supere a economia obtida com os limitantes melhores. É importante notar que os limitantes derivados de $PMM_{\bar{\lambda}}$ e $PMM_{\bar{\pi}}$ são sempre melhores ou iguais aos obtidos por PMM_{L}.

2.3.2 - PROCEDIMENTOS DE REDUÇÃO E MÉTODOS DE SOLUÇÃO

O tamanho do problema PMM pode ser reduzido, de uma maneira similar à apresentada na seção 2.1.2 através da construção dos conjuntos:

$$I1 = \{i : \sum_{j=1}^{m} y_{ji} = 1 \text{ é uma solução ótima para PMM}\}$$

$$I0 = \{i : \sum_{j=1}^{m} y_{ji} = 0 \text{ é uma solução ótima para PMM}\}$$

No presente caso no entanto, apenas os elementos do conjunto I0 permitem a redução do tamanho do problema (eliminando os itens cujos índices pertencem a I0), I1 fornece informação útil apenas para reduzir o número de nós da árvore de decisão em um algoritmo do tipo particionar e limitar, uma vez que ele não especifica em qual mochila o item foi inserido. Ingargiola e Korsh [IK] apresentam um procedimento baseado na extensão da relação de dominância entre os itens para construir os conjuntos I1 e I0.

São esboçados a seguir dois algoritmos para encontrar a solução exata do problema.

O primeiro algoritmo, prôposto por Martelo e Toth [MT3], é baseado no método particionar e limitar. A cada nó é resolvido a relaxação lagrangeana do problema com $\lambda_i = 0$ para todo i, isto é, são resolvidos m problemas da mochilas 0/1 independentes. Se nenhum item aparece em duas ou mais mochilas, foi encontrada uma solução factível para o problema e o movimento regressivo é efetuado. Senão, um item que aparece em m' mochilas ($2 \leq m' \leq m$) é selecionado e são gerados m' nós descendentes correspondentes a inserção deste item nas m'-1 primeiras mochilas e o m'-ésimo nó corresponde a sua exclusão delas. Os m' limitantes superiores são

calculados resolvendo m' problemas da mochila 0/1 e usando parte das soluções encontradas previamente nos nós anteriores.

Em Martello e Toth [MT2] é utilizada uma técnica particular de busca em árvore chamada "bound and bound" que a cada nó da arvore de decisão calcula um limitante superior e um limitante inferior para o problema corrente. O limitante superior sup(S) para o problema corrente é calculado através da relaxação por substituição de S e o limitante inferior inf(S) é calculado por um procedimento heurístico que encontra uma solução para a primeira mochila, exclui os itens pertencentes a esta solução, encontra a solução para a segunda e assim por diante. A solução heurística é armazenada em $\hat{y}_{ji}$. A cada iteração, a partição do problema é feita tomando a próxima variável y_{ji} tal que $\hat{y}_{ji} = 1$, de acordo com valores crescentes de j. A principal diferença entre este procedimento e o procedimento particionar e limitar padrão é que a fase de partição do problema é feita aqui adaptando a solução parcial através da solução obtida no cálculo dos limitantes inferiores. Isto fornece duas vantagens importantes: 1) para todo problema corrente S com sup(S) = inf(S), $\hat{y}$ é uma solução ótima; 2) para S com inf(S) < sup(S), S é adaptado através da solução heurística que é melhor que a solução obtida por uma série de movimentos progressivos adicionais.

Os algoritmos para a solução exata de PMM podem ser aplicados, com um tempo computacional aceitável, apenas para problemas com valor de m pequeno. No entanto, as aplicaçõoe práticas deste problema geralmente envolvem um grande número de variáveis, exigindo assim o uso de métodos de solução mais adequados: heurísticas.

Fisk e Hung [FH] apresentam um procedimento heurístico baseado na solução exata da relaxação de substituição para PMM. Esse procedimento tenta inserir os itens do conjunto $\mathbb{RS}$ ($\mathbb{RS} = \{i : y_{ji} = 1, y_{ji}$ solução de $PMM_{\pi}\}$ nas mochilas. Quando um item não puder ser inserido em nenhuma delas, tenta-se fazer troca de itens entre cada par de mochilas (um por um, um por dois, dois por um ...) até ser encontrada uma troca que utilize totalmente o espaço diponível em uma delas. Se todos os itens do conjunto $\mathbb{RS}$ foram usados, uma solução ótima foi encontrada; caso contrário a solução factível corrente pode ser melhorada inserindo nas mochilas o maior número possível de itens do conjunto $\mathbb{N} - \mathbb{RS}$. Como este algoritmo requer a solução exata para a relaxação de substituição do problema PMM, ele requer no pior caso um tempo de execução não polinomial.

Martello e Toth [MT4] apresentam um procedimento heurístico baseado no procedimento apresentado em Fisk e Hung [FH]. O algoritmo consiste de três etapas: 1) determinar uma solução inicial preenchendo cada mochila com itens consecutivos, iniciando com a mochila com menor capacidade; 2) rearranjar a solução factível encontrada fazendo troca de itens entre as mochilas de forma que cada uma contenha itens com razão valor/peso diferentes; 3) melhorar o rearranjo da solução encontrada, e em seguida tentar excluir cada item da solução corrente e trocá-lo por um ou mais itens que não estavam na solução e que levam a um melhoramento da mesma. Os experimentos computacionais feitos com este procedimento indicam que ele tem um comportamento satisfatório tanto em termos de tempo de execução como em qualidade da solução encontrada.

2.4 - PROBLEMA DO NÚMERO MÍNIMO DE MOCHILAS [EC, JDUGG]

Considere agora, m mochilas com a mesma capacidade, C, e n itens com peso p_i (i = 1..n). O problema do Número Mínimo de Mochilas consiste em determinar como distribuir estes itens entre as mochilas de forma a não exceder suas capacidades e utilizando o menor número possível delas. Em geral, supõe-se que o peso dos itens está dentro do intervalo (0, 1] e a capacidade das mochilas é C = 1 . Este problema é um caso especial do problema do Corte Unidimensional (cap. III). Modela diversos problemas práticos na ciência da computação. Alguns exemplos são:

FORMATAÇÃO - Suponha que as mochilas sejam registros de tamanho fixo: k bits, e os itens sejam dados requerendo kp_1, kp_2, ..., kp_n bits. O objetivo é alocar os dados usando o menor número de registros.

PAGINAÇÃO - Aqui as mochilas são páginas e os itens são segmentos de um programa que devem aparecer em uma única página, por exemplo: loops, arrays, etc.

ALOCAÇÃO DE ARQUIVOS - O objetivo é alocar arquivos de vários tamanhos no menor número possível de trilhas no disco observando que os arquivos não podem ser divididos entre trilhas.

Serão apresentados aqui um método exato baseado na formulação do problema em programação inteira 0/1 e quatro heurísticas clássicas para o problema.

2.4.1 - MÉTODO EXATO

O método exato descrito a seguir foi proposto em Eilon e Christofides [EC], e soluciona o problema formulado matemáticamente da seguinte maneira:

$$\min \quad \sum_{j=1}^{m} \left(h_j \sum_{i=1}^{n} y_{ji} \right)$$

s.a

$$\sum_{i=1}^{n} p_i \, y_{ji} \leq C \qquad j = 1..m$$

$$\sum_{j=1}^{m} y_{ji} \leq 1 \qquad i = 1..n$$

$$y_{ji} = 0/1 \qquad i = 1..n$$

$$\text{e } j = 1..m$$

onde:

p_i = peso do item i

y_{ji} = 0(1) indica que o item i foi excluído (incluído) da mochila j

h_j = penalidade atribuída ao se alocar itens na mochila j, $h_j < h_{j+1}$

O algoritmo trabalha encontrando inicialmente um limitante superior z' para o problema por inspeção ou usando uma heurística.

Do conjunto de nxm variáveis é formado um subconjunto S de k variáveis, cada uma com valor 0 ou 1. As outras (nxm - k) variáveis ficam livres. O conjunto S representa então um subproblema com (nxm - k) variáveis, observe que como $h_j > 0$, o melhor valor (apesar de não necessáriamente factível) para as variáveis livres é 0. Este subproblema está resolvido quando é determinado o melhor valor das variáveis livres (estes valores

juntamente com os valores do elementos do conjunto S, formam uma solução factível), ou quando nenhuma solução factível é obtida. No primeiro caso, a solução é guardada para futuras referências.

Quando uma solução y não é factível, é definido um subconjunto T com todas as variáveis livres correspondentes a nós que não violam a capacidade da mochila e o limitante z'. Então um membro do conjunto T é transferido para o conjunto S e um novo subproblema aparece. Neste ponto, é usado uma formulação do subproblema em programação linear (PL) de forma a excluir alguns nós da árvore de decisão. Se a solução do PL excede z', ou se a solução é infactível, o nó é excluído.

Quando a solução é factível, verifica-se no conjunto S se há alguma variável que mudada para seu complemento produz um subproblema que ainda não foi resolvido. Se isto ocorre, esta variável troca de valor, as variáveis seguintes tornam-se livres e o algoritmo prossegue. Caso contrário, y é a solução ótima.

O método exposto acima apresenta um desempenho em termos de tempo computacional muito ruim, justificando a fato de que na prática são usados algoritmos aproximados para o problema.

2.4.2. - MÉTODOS HEURÍSTICOS

Como foi dito antes, algoritmos que buscam a solução exata para este problema requerem um geral uma busca combinatória muito lenta, e os algoritmos heurísticos tratados na prática enontram soluções muito próximas da ótima em um tempo computacional muitas vezes menor. São apresentados a seguir quatro algoritmos heurísticos clássicos.

Algoritmo 1 - (First Fit) - Sejam as mochilas $C_1, C_2, \ldots, C_m$ cada uma com nível de ocupação inicial zero, e os itens $p_1, p_2, \ldots, p_n$ na ordem original. Para alocar o item p_i encontre o menor índice j tal que a mochila C_j esteja preenchida até o nível $\beta \leq 1 - p_i$ e coloque p_i em C_j. O nível de C_j é então mudado para $\beta + p_i$.

Algoritmo 2 - (Best Fit) - Sejam as mochilas $C_1, C_2, \ldots, C_m$ cada uma com nível de ocupação inicial zero, e os itens $p_1, p_2, \ldots, p_n$ na ordem original. Para alocar o item p_i encontre o menor índice j tal que o nível da mochila C_j mude para $\beta \leq 1 - p_i$ e β seja o maior possível. Coloque p_i em C_j e mude o nível de C_j para $\beta + p_i$.

Algoritmo 3 - (First Fit Decreasing) - Ordene o conjunto de itens de forma não crescente e aplique o Algoritmo 1.

Algoritmo 4 - (Best Fit Decreasing) - Ordene o conjunto de itens de forma não crescente e aplique o Algoritmo 2.

Em Jonhson et alli [JDUGG] foi demonstrada uma análise do pior caso baseada no máximo valor, $R_F(k)$, da razão $F(n)/M_o$ sobre todos os conjuntos de itens para os quais $M_o = k$, onde $F(n) \in (FF(n), BF(n), FFD(n), BFD(n))$ é usado para denotar o número de mochilas obtidas aplicando os quatro últimos algoritmos, respectivamente, a um conjunto de n itens e M_o é um limitante para o problema:

$$\sum_{i=1}^{n} p_i \;/\; C \leq M_o \leq \left(\sum_{i=1}^{n} p_i \;/\; C \right) + 1$$

Os autores chegaram aos seguintes resultados:

$$\lim_{k\to\infty} R_{FF}(k) = 17/10.$$

$$\lim_{k\to\infty} R_{BF}(k) = 17/10$$

$$\lim_{k\to\infty} R_{FFD}(k) = 11/9$$

$$\lim_{k\to\infty} R_{BFF}(k) = 11/9.$$

CAPÍTULO III

3 - O PROBLEMA DO CORTE UNIDIMENSIONAL

Este capítulo trata do problema do corte onde apenas uma dimensão (largura ou comprimento) da peça de tamanho padrão é considerada para o corte. Nas próximas seções são apresentadas formulações do Problema Unidimensional, métodos de solução e algumas variações do mesmo.

3.1 - FORMULAÇÃO

O Problema do Corte Unidimensional consiste em determinar como cortar o menor número de peças de tamanho padrão L, atendendo a uma demanda, b_i, de itens de tamanho l_i, $i = 1..\ n$, supondo que existe uma quantidade ilimitada de peças de tamanho padrão.

Sendo x_j o número de vezes que o esquema de corte A_j é usado, o problema consiste em determinar o vetor x (m x 1) solução de:

$$\min \{ \sum_{j=1}^{m} x_j : Ax \geq b,\ x \in \mathbb{Z}_+^m \} \qquad (3.1)$$

onde:

$b = (b_i)$ vetor (n x 1) de demanda.

$A = (a_{ij})$ matriz (n x m) onde cada coluna A_j representa um esquema de corte com a_{ij} igual ao número de itens de dimensões l_i que são obtidas neste esquema.

3.2 - MÉTODOS DE SOLUÇÃO

Esta formulação do problema apresenta duas grandes dificuldades: a restrição aos inteiros e o grande número de esquemas de corte (colunas da matriz A).

Uma maneira de diminuir estas dificuldades é relaxar a restrição $x \in \mathbb{Z}_+^n$ em $x \in \mathbb{R}_+^n$, e resolver o programa linear resultante através do método simplex revisado gerando a cada iteração uma coluna candidata para entrar na base (seção 1.2.2).

Como vimos na seção 1.3.2, a coluna candidata a entrar na base, A_e, deve satisfazer:

$$\bar{c}_e = \min_j \{ \bar{c}_j \}$$

isto é:

$$\bar{c}_e = \min_j \{ 1 - \pi A_j \}$$

além disso, o vetor A_j, deve descrever um esquema de corte e portanto :

$$a_{ij} \in \mathbb{Z}_+$$

$$\sum_{i=1}^{n} l_i a_{ij} \leq L \ .$$

Assim, o problema de encontrar a coluna com menor custo relativo se torna:

$$\max \sum_{i=1}^{n} \pi_i a_i$$

$$\text{s. a} \quad \sum_{i=1}^{n} l_i \, a_i \leq L$$

$$a_i \in \mathbb{Z}_+$$

que como vimos na seção 2.2 é o Problema da Mochila Inteiro.

3.3.- VARIAÇÕES DO PROBLEMA

Além do objetivo apresentado na seção anterior:

i) minimizar o número total de peças de tamanho padrão:

$$\min \sum_{j=1}^{m} x_j$$

podem ser associadas ao Problema do Corte Unidimensional ainda, diferentes funções objetivos, dentre as quais:

ii) minimizar a perda total :

$$\min \sum_{j=1}^{m} r_j x_j$$

onde r_j é a perda em um esquema de corte A_j dada por:

$$r_j = L - \sum_{i=1}^{n} l_i \, a_{ij} \, . \qquad (3.2)$$

iii) minimizar a perda relativa:

$$\min \left(\sum_{j=1}^{m} r_j x_j \;/\; \sum_{j=1}^{m} x_j \right)$$

iv) maximizar o valor total dos itens cortados:

$$\max \left(\sum_{i=1}^{n} \sum_{j=1}^{m} v_i \; a_{ij} \; x_j \right)$$

onde v_i é um valor atribuído ao item i.

Diferentes conjuntos de restrições são associados a estas funções objetivos, os mais comuns são:

i) $Ax = b,\ x \in \mathbb{Z}_+^n$

ii) $Ax \geq b,\ x \in \mathbb{Z}_+^n$

iii) $Ax \leq b,\ x \in \mathbb{Z}_+^n$

iv) $b' \leq Ax \leq b,\ x \in \mathbb{Z}_+^n$

Observe que os problemas formados pela restrição i) combinada com a função objetivo i) ou com a função objetivo ii) são equivalentes.

Na indústria do corte de papel, frequentemente as encomendas são atendidas dentro de um intervalo de tolerância de mais ou menos 5% (Gilmore-Gomory [GG2]). Isto é, $b' \leq Ax \leq b$. Neste caso, afim de aproveitar esta tolerância, a função objetivo mais apropriada é minimizar a perda relativa, pois se forem usados os objetivos i), ii) ou iv) acima a solução ótima tende a satisfazer o nível mínimo de produção.

Assim, se um esquema de corte A_j é usado x_j vezes, o problema é:

$$\min \frac{z_1}{z_2} = \left(\frac{\sum_{j=1}^{m} r_j x_j}{\sum_{j=1}^{m} x_j} \right)$$

$$\text{s.a} \qquad Ax \geq b' \qquad (3.5)$$

$$Ax \leq b \qquad (3.6)$$

$$x \in \mathbb{Z}_+^n$$

acrescentando as variáveis de folga em (3.6) e subtraindo a equação resultante de (3.5) chega-se ao problema:

$$\min \frac{z_1}{z_2}$$

$$\text{s. a} \quad Ax + s = b$$

$$0 \leq s \leq b' - b$$

$$x \in \mathbb{Z}_+^n$$

que é um problema de Programação Fracionária com variáveis de folga canalizadas.

Observe que $\sum_{j=1}^{m} x_j > 0$ para qualquer vetor de demanda $b > 0$, e que como os elementos da matriz A são todos números inteiros positivos o conjunto de restrições deste problema forma um conjunto limitado. Ou seja, este problema satisfaz as condições do

teorema 1.2.1, (seção 1.2.4) e portanto pode ser resolvido pelo método simplex com os custos relativos calculados pala equação (1.11) (seção 1.2.4).

Observe ainda que a canalização das variáveis de folga pode ser trabalhada sem aumentar o tamanho da base do problema, usando o método simplex para variáveis canalizadas (seção 1.2.3).

A única dificuldade na resolução do problema está então no número alto de colunas da matriz A. Usando o algoritmo simplex da seção 1.2.1 para resolver o problema, a escolha da variável candidata a entrar na base requer que os custos relativos sejam calculados pela equação (1.11), isto é:

$$\bar{c}_j = \frac{\bar{z}^2 \; \bar{c}_j^{\,1} - \bar{z}^1 \; \bar{c}_j^{\,2}}{(\bar{z}^2)^2}$$

mas, pelas equações (1.9) e (1.10):

$$\bar{c}_j^{\,1} = c_j^1 - \pi^1 A_j = r_j - \pi^1 A_j \qquad (3.7)$$

$$\bar{c}_j^{\,2} = c_j^2 - \pi^2 A_j = 1 - \pi^2 A_j \qquad (3.8)$$

onde π^1 e π^2 são soluções dos sistemas: $\pi^1 B = c_B^1$ $\quad$ $\pi^2 B = c_B^2$

substituindo (3.7) e (3.8) em (1.11):

$$\bar{c}_j = \frac{\bar{z}^2 \; (r_j - \pi^1 A_j) - \bar{z}^1 \; (1 - \pi^2 A_j)}{(\bar{z}^2)^2} \qquad (3.9)$$

substituindo o valor de r_j (eq. 3.2) em (3.9):

$$\bar{c}_j = \frac{\bar{z}^2 \left[\left(L - \sum_{i=1}^{n} l_i \, a_{ij} \right) - \pi^1 A_j \right] - \bar{z}^1 \left(1 - \pi^2 A_j \right)}{\left(\bar{z}^2 \right)^2}$$

$$\bar{c}_j = \frac{\left(\bar{z}^2 L - \bar{z}^1 - \bar{z}^2 \sum_{i=1}^{n} l_i \, a_{ij} - \bar{z}^2 \sum_{i=1}^{n} \pi_i^1 \, a_{ij} - \bar{z}^1 \sum_{i=1}^{n} \pi_i^2 \, a_{ij} \right)}{\left(\bar{z}^2 \right)^2}$$

$$\bar{c}_j = \frac{\left(\bar{z}^2 \, L - \bar{z}^1 - \sum_{i=1}^{n} \left[\bar{z}^1 \, \pi_i^2 + \bar{z}^2 \left(l_i + \pi_i^1 \right) \right] a_{ij} \right)}{\left(\bar{z}^2 \right)^2}$$

fazendo: $k = \bar{z}^2 \, L - \bar{z}^1$ e $\pi_i = \bar{z}^1 \, \pi_i^2 + \bar{z}^2 \left(l_i + \pi_i^1 \right)$

$$\bar{c}_j = \frac{k - \sum_{i=1}^{n} \pi_i \, a_{ij}}{\left(\bar{z}^2 \right)^2} .$$

Assim, se for usado o critério de Dantzig para escolher a candidata a entrar na base, tem-se que:

$$\min_j \bar{c}_j = \frac{\min_j \left(k - \sum_{i=1}^{n} \pi_i \, a_{ij} \right)}{\left(\bar{z}^2 \right)^2}$$

que é equivalente a :

$$\max \sum_{i=1}^{n} \pi_i \, a_i$$

$$\text{s. a} \sum_{i=1}^{n} l_i \, a_i \leq L$$

$$a_i \in \mathbb{Z}_+$$

Ou seja, novamente, para gerar uma coluna candidata a entrar na base é resolvido um problema da mochila inteiro. A mudança aqui em relação ao método descrito na seção 3.2.1, é que π_i é função de π_i^1 e π_i^2. O cálculo do custo relativo das variáveis de folga é feito diretamente pela equação (1.11).

3.4 - OUTRAS VARIAÇÕES DO PROBLEMA

Freqüentemente, o número de itens que podem ser obtidos pelo corte de uma peça de tamanho padrão é limitado pelo número de facas disponíveis. Gilmore e Gomory [GG2] resolvem este problema pelo método simplex com geração de colunas (seção 3.2.1) acrescentando ao problema da mochila a restrição:

$$\sum_{i=1}^{n} a_i \leq F$$

onde F é o número de facas disponíveis.

Haesler [Has1] considera no problema do corte além das perdas, o custo pela mudança do esquema de corte. Formula então o problema atribuindo uma penalização para mudanças no esquema de corte e propõe um algoritmo heurístico para encontrar a solução ótima que minimiza as perdas e o número de esquemas de corte necessários para atender a demanda. Em [Has], Haesler observa que esse procedimento heurístico não trabalha bem quando o tamanho das peças da demanda são grandes em relação ao tamanho da peça padrão. Nestes casos, o procedimento de programação linear apresentado em Gilmore e Gomory [GG2] continua sendo uma boa ferramenta para

encontrar uma solução. Basta limitar o número máximo de vezes que uma peça i da demanda pode aparecer em um esquema de corte. Esta restrição faz com que um determinado esquema seja usado mais vezes e portanto ao fazer o arredondamento para valores inteiros o número de esquemas de corte diferentes e o número de itens da demanda não atendidos é menor.

Ferreira [Fe] modifica o algoritmo apresentado em Gilmore e Gomory [GG2] para resolver o problema com restrições adicionais representando cortadeiras e guilhotinas. Com seu modelo, determina esquemas de corte para dividir bobinas de papel de tamanho-padrão L em bobinas menores de tamanho $l_i \leq L$, e bobinas de tamanho l_i em folhas de dimensões $(\alpha_i \times \beta_i)$.

3.5 - OUTRAS FORMULAÇÕES

Dyckhoff [Dy] propõe um modelo onde os itens são obtidos pela divisão da peça de corte (peças de tamanho padrão) usando apenas uma faca, em um número ilimitado de vezes, e onde os itens residuais (itens cortados que não fazem parte da demanda) constituem novas peças de corte. Desta maneira, um esquema de corte é representado por uma estrutura do tipo: [k, l]; isto é, a peça de tamanho padrão k é cortada na posição l, onde l é o tamanho de um item da demanda, produzindo duas peças de tamanho l e k-l. Apesar desta estrutura simples, qualquer esquema de corte pode ser construído sucessivamente. A peça de tamanho k-l, se não fizer parte da demanda, e se k-l > min (l_i), será considerada como uma nova peça de corte e denominada item residual relevante.

Baseado nesta nova estrutura para um esquema de corte, o modelo é:

Dados,

$\mathbb{S} = \langle S_1 \,..\, S_k \rangle$, conjunto das peças de tamanho padrão em estoque.

c_i = custo da peça de tamanho padrão S_i.

$\mathbb{L} = \langle l_1 \,..\, l_n \rangle$, tamanho dos itens da demanda.

$b = \langle b_1 \,..\, b_n \rangle$, demanda das peças de tamanho l_i

$\mathbb{R} = \langle r_1 \,..\, r_r \rangle$, tamanho dos itens residuais relevantes

$\min \langle r_i \rangle \geq \min \langle l_i \rangle$

deseja-se determinar:

y_{kl} = número de peças de tamanho padrão k (originais ou residuais) cortadas na posição l, $l \in \mathbb{L}$, $l \leq k$, que minimizem os custos satisfazendo a demanda. Isto é:

$$\min \quad \sum_{l \in \mathbb{S}} c_l \left(\sum_{k \in \mathbb{C}^l} y_{lk} - \sum_{k \in \mathbb{B}^l} y_{k+l,k} \right) \qquad (3.2)$$

$$\text{s. a} \quad \sum_{k \in \mathbb{A}^l} y_{kl} + \sum_{k \in \mathbb{B}^l} y_{k+l,k} \geq \sum_{k \in \mathbb{C}^l} y_{lk} + b_l \qquad (3.3)$$

$$l \in \langle \mathbb{L} \cup \mathbb{R} \rangle \setminus \mathbb{S}$$

$$y_{kl} \in \mathbb{Z}^+$$

onde:

$\mathbb{S} \cup \mathbb{R}$ = conjunto das peças de corte

$\mathbb{A}^l = \{k \in \mathbb{S} \cup \mathbb{R} : k > l\}$

$\mathbb{B}^l = \{k \in \mathbb{L} : k + l \in \mathbb{S} \cup \mathbb{R}\}$

$\mathbb{C}^l = \{k \in \mathbb{L} : k < l\}$

$$b_l = \begin{cases} b_i \,, & l \in \mathbb{L} \text{ e } l = l_i \\ 0 \,, & l \notin \mathbb{L} \end{cases}$$

A função objetivo (3.2), indica que o custo a ser minimizado é dado pelo número de peças de tamanho padrão originais usadas, subtraídas do número de itens residuais relevantes obtidos pelo corte desta peça e utilizados para obter itens da demanda. As equações de balanceamento (3.3), indicam que o número de peças de corte (originais ou residuais) disponíveis deve ser maior ou igual ao número de peças de tamanho l cortada ($l \in \mathbb{L} \cup \mathbb{R}$). Não há necessidade de escrever estas equações para $l \in \mathbb{S}$, porque as peças deste tamanho estão disponíveis em qualquer quantidade.

Comparando os dois modelos, Dyckhoff [Dy] observa que apesar dos dois modelos serem equivalentes, eles diferem nas dimensões. O número de restrições na nova formulação cresce do número de itens da demanda, para o número de itens residuais relevantes diferentes das peças de tamanho padrão, somado ao número de itens da demanda. O número de variáveis decresce do número de esquemas de corte associados a cada peça de tamanho padrão, para o número de possiblidades de cortar uma vez as peças de tamanho padrão ou itens residuais relevantes. É impossível dar estimativas válidas universalmente para as dimensões dos dois modelos uma vez que eles não dependem apenas do número de tamanhos padrões e do número de itens da demanda, mas também das relações entre os tamanhos dos itens.

Dyckhoff observa ainda que a nova formulação é melhor em termos de esforço computacional se existem várias peças de tamanhos padrões diferentes, e/ou o conjunto de itens residuais relevantes não tiver cardinalidade muito superior ao número de itens da demanda.

Esta formulação possui um número menor de variáveis do que a

formulação anterior mas possui um número maior de restrições. No entanto, relaxando a restrição das variáveis assumirem valores inteiros, de acordo com Dyckhoff, é possível resolver os problemas encontrados na prática pelo método simplex usual.

É bom observar também, que na prática este modelo só funciona quando não existe restrição no número de facas.

Finalmente, deve-se ressaltar que o problema do número mínimo de mochilas apresentado no capítulo II corresponde ao seguinte modelo do corte unidimensional:

$$\min \{ \sum_{j=1}^{n} x_j : Ax = b,\ x \in \mathbb{Z}_+^n \}$$

CAPÍTULO IV

4 - O PROBLEMA DO CORTE BIDIMENSIONAL

O Problema do Corte Bidimensional aparece no processo de produção onde uma placa retângular deve ser cortada em peças menores. Duas dimensões, usualmente comprimento e largura, são consideradas contrastando com o problema descrito no capítulo anterior onde somente uma dimensão é considerada.

Encontram-se aplicações deste problema no corte de placas de aço, vidro, papel, filme, plástico, couro, tecido, madeira e na diagramação de jornais e revistas; assim como no carregamento de caminhões e vagões de trens, embora este tipo de corte possa ser considerado tri ou quadri-dimensional (comprimento, largura, altura e peso).

O Problema do Corte Bidimensional com maior aplicação prática é o que considera o corte de uma placa retângular padrão em itens retângulares e tem recebido atenção de diversos autores com diferentes formulações e métodos de resolução.

O presente capítulo apresenta diferentes formulações e métodos de resolução. É dado destaque às técnicas de geração de colunas e são apresentados ainda variações do problema enfocando aspectos práticos que aparecem na indústria.

4.1 - FORMULAÇÃO

O Problema do Corte Bidimensional (CB) consiste em determinar esquemas para cortar o menor número de placas retângulares de tamanho padrão L x W de forma a atender uma demanda, b_i, de itens retângulares de dimensões l_i x w_i, i = 1..n.

$$CB : \min \{ \sum_{j=1}^{m} x_j : Ax \geq b, x \in \mathbb{Z}_+^n \}$$

onde:

$x = (x_j)$ é um m-vetor com cada componente representando o número de vezes que o esqema de corte j é utilizado para o corte de uma placa.

$A = (a_{ij})$ é uma (n x m)-matriz onde cada coluna A_j representa um esquema de corte e a_{ij} é o número de itens de dimensões l_i x w_i que serão obtidos no esquema A_j.

$b = (b_i)$ é um n-vetor de demanda dos itens retangulares.

A exemplo do Problema do Corte Unidimensional, além do objetivo acima:

i) minimizar o número total de peças de tamanho padrão:

$$\min \sum_{j=1}^{m} x_j$$

podem ser associadas a este problema ainda, diferentes funções objetivos dentre as quais:

ii) minimizar a perda total :

$$\min \sum_{j=1}^{m} r_j x_j$$

onde r_j é a perda em um esquema de corte A_j dada por:

$$r_j = ((L \times W) - [\sum_{i=1}^{n} a_{ij} (l_i * w_i)])$$

iii) minimizar a perda relativa:

$$\min (\sum_{j=1}^{m} r_j x_j \; / \; \sum_{j=1}^{m} x_j)$$

iv) maximizar o valor total dos itens cortados:

$$\max (\sum_{i=1}^{n} \sum_{j=1}^{m} v_i \; a_{ij} \; x_j)$$

onde v_i é um valor atribuído ao item i.

Diferentes conjuntos de restrições são associados a estas funções objetivos, os mais comuns são:

i) $Ax = b, \; x \in \mathbb{Z}_+^n$

ii) $Ax \geq b, \; x \in \mathbb{Z}_+^n$

iii) $Ax \leq b, \; x \in \mathbb{Z}_+^n$

iv) $b' \leq Ax \leq b, \; x \in \mathbb{Z}_+^n$

4.2 - MÉTODOS DE SOLUÇÃO

Uma maneira de resolver CB é utilizar a programação linear. Para isso a restrição $x \in \mathbb{Z}_+^n$ é relaxada em :

$$x \in \mathbb{R}_+^N .$$

Como no problema do corte unidimensional descrito anteriormente, a dificuldade de resolver este problema consiste no número enorme de colunas da matriz A. Se, como no caso anterior, for usado o método Simplex com de geração de colunas, a cada iteração o seguinte problema deve ser resolvido:

$$GC : \max \{ \pi y : y \in \{ A_1 .. A_n \} \text{ (y um esquema de corte) } \}$$

Ou seja, trata-se do problema da corte bidimensional com apenas uma placa disponível para o corte e com objetivo de maximizar o valor total dos itens cortados. Em contraste com o Problema do Corte Unidimensional, é difícil encontrar uma descrição matemática para a restrição $y = A_j$ (esquema de corte).

Diversos autores têm proposto métodos de solução para este problema. A melhor solução para o problema CB relaxado vai depender diretamente da qualidade da solução de GC. Na seção 4.3 são desenvolvidos alguns métodos para solucionar GC.

Para resolver o problema de minimizar a perda total (função objetivo ii) da seção 4.1) Wang [W] apresenta um método aproximado que consiste em num primeiro momento gerar as colunas da matriz A, ou seja , os esquemas de corte , e rejeitar aqueles que possuem

uma perda de corte maior que um determinado valor pré-fixado, por exemplo, uma porcentagem da área da placa. Em seguida, aplicar o método simplex usual (seção 1.2.1) às colunas restantes.

4.3 - GERAÇÃO DOS ESQUEMAS DE CORTE

Gerar as colunas da matriz A para resolver CB é equivalente a encontrar uma solução para o problema GC. Ao gerar os esquemas de corte, um fato a ser considerado é o tipo de corte a ser feito. Um corte é do tipo guilhotina quando é efetuado de uma aresta à aresta oposta do retângulo, paralelo às duas arestas restantes. Diversas aplicações do problema CB, como por exemplo na indústria de vidro, papel, madeira, etc, exigem este tipo de corte. Cortes não guilhotinados ocorrem com mais frequência quando nenhum corte é realmente efetuado, como na alocação de anúncios em jornais ou no carregamento de caminhões e trens.

Um corte pode ser feito ao longo do comprimento, paralelo à largura, chamado corte vertical ou ao longo da largura, paralelo ao comprimento, corte horizontal.

O número de esquemas de corte factíveis pode ser reduzido se as seguintes propriedades forem consideradas:

i) CORTE NORMAL OU CANÔNICO - Apenas os cortes na placa que são múltiplos inteiros do comprimento ou da largura total de uma ou mais peças devem ser considerados. Ou seja, se é feito um corte vertical no retângulo de dimensões p x q na posição $\bar{p}$, então, no modelo de corte final, deve haver alguma combinação de itens com

comprimento l_i e largura $w_i \leq q$, cuja soma total dos comprimentos deve ser exatamente $\bar{p}$. Se isso não ocorre, então um corte alternativo em posição menor ou igual a $\bar{p}$ pode ser feito levando ao mesmo esquema de corte final, por isso de mesmo valor, de ocordo com a condição descrita acima. Um corte normal envolvendo apenas um tipo de retângulo é chamado corte homogêneo.

ii) EFEITO DE SIMETRIA - Com um corte vertical no retângulo de dimensões p x q na posição p', obtem-se dois retângulos de dimensões p'x q e (p-p') x q. Claramente, estes dois retângulos poderiam ser obtidos com um corte em p-p'. Essa duplicação de esquemas de corte pode ser evitada, para todo corte canônico não-homogêneo da placa, restringindo os cortes a posições menores ou iguais à metade da correspondente dimensão da placa. Isso reduz o número de possibilidades de cortes pela metade.

iii) EFEITO DE ORDENAÇÃO - Suponha que é feito um corte vertical no retângulo de dimensões p x q na posição p' produzindo dois retângulos de dimensões p'x q e (p - p') x q. Em seguida considere que no retângulo de dimensões (p - p') x q é feito um corte na posição p'', onde p'< p'' $\leq \lfloor$(p - p') / 2$\rfloor$, produzindo três retângulos: (p',q), (p'',q), (p - p'- p'',q). Esses mesmos três retângulos poderiam ter sido obtidos fazendo o corte p'' em (p,q) e em seguida o corte p' em (p -p'',q). Esse tipo de duplicação é evitado introduzindo uma ordenação nos cortes tal que se um retângulo é cortado na posição p', então todos os cortes subsequentes nos dois retângulos resultantes devem ser maiores ou iguais a p'.

A seguir são apresentadas algumas técnicas de geração de colunas considerando o corte do tipo guilhotina.

4.3.1 - MÉTODO EM DOIS ESTÁGIOS [GG3]

Uma importante subclasse de esquemas de corte do tipo guilhotina é aquela onde os cortes são efetuados em apenas dois estágios. Primeiro a placa (L, W) é dividida em faixas de tamanho L x w_i e então cada faixa é tomada individulamente e cortada em seu comprimento. Às vezes um estágio de ajuste é permitido e os itens obtidos no 1° e no 2° estágio são cortados na largura para produzir os itens da demanda.

Assim, supondo que os itens estão ordenados em ordem crescente de largura ($w_1 \leq w_2 \leq \ldots \leq w_n$), gerar os esquemas de corte em dois estágios consiste em :

1) Para todas as larguras w_i (i = 1..n) calcula-se π_i^* , o valor ótimo obtido alocando os itens de tamanho l_j x w_j ,para $w_j \leq w_i$, lado a lado em uma faixa de comprimento L e largura w_i. É resolvido então para cada i o seguinte problema da mochila:

$$\begin{aligned} \pi_i^* = \quad & \max \quad \pi_1 s_{i1} + \ldots + \pi_k s_{ik} \\ & s.a \quad l_1 s_{i1} + \ldots + l_k s_{ik} \leq L \\ & \qquad s_{ij} \in \mathbb{Z}_+ \quad e \; w_j < w_i \quad j = 1..k \end{aligned}$$

2) A solução ótima para GC é obtida resolvendo mais um problema da mochila:

$$\begin{aligned} \max \quad & \pi_1^* t_1 + \ldots + \pi_n^* t_n \\ s.a \quad & w_1 t_1 + \ldots + w_n t_n \leq W \\ & t_i \in \mathbb{Z}^+ \end{aligned}$$

O esquema de corte j é então:

$$a_{ij} = \sum_{k=1}^{n} t_k \, s_{ik} .$$

Existem então m+1 problemas da mochila para serem resolvidos. No entanto, é possível resolver os m problemas da mochila que aparecem no estágio 1) juntos, através do procedimento de programação dinâmica descrito na seção 2.2.3.

Portanto resolver o problema GC em dois estágios consiste então em primeiro calcular o valor correspondente a se alocar itens em faixas de comprimento L e largura w_i (considerando apenas os itens com largura $w_j \leq w_i$), e em seguida determinar a combinação dessas faixas que cabem dentro da largura W e que maximizam o valor total do esquema de corte. Em resumo, é necessário resolver apenas dois problemas da mochila.

4.3.2 - MÉTODO EM N-ESTÁGIOS [Be1]

O método apresentado nessa seção trabalha com a restrição ao problema que limita os cortes a serem feitos em um número n de estágios.

Associa-se a cada estágio um tipo de corte, horizontal ou vertical. É importante notar, no entanto, que não é exigido que um corte seja efetuado a cada estágio mas, sim que a direção de corte seja mudada.

Para formular o problema define-se as seguintes equações:

F(k, x, y) = valor ótimo do corte no k-ésimo estágio do item de dimensão (x, y) considerando que o corte no 1º estágio foi horizontal.

G(k, x, y) = valor ótimo do corte no k-ésimo estágio do item de dimensão (x, y) considerando que o corte no 1º estágio foi vertical .

O valor k = 0 na definição acima corresponde a um estágio adicional de corte para ajustar um item no retângulo (x, y). São identificados quatro casos:

1) (T = 1) Não é possível um estágio adicional.

$$F(0, x, y) = \max \{ \{0\} \cup \{v_i : l_i = x , w_i = y, i = 1..n \} \}$$

2) (T = 2) É possível um estágio adicional com um corte horizontal.

$$F(0, x, y) = \max \{ \{0\} \cup \{v_i : l_i = x , w_i \leq y, i = 1..n \} \}$$

3) (T = 3) É possível um estágio adicional com um corte vertical.

$$F(0, x, y) = \max \{ \langle 0 \rangle \cup \langle v_i : l_i \leq x \ , w_i = y, i = 1..n \rangle \}$$

4) (T = 4) É possível um estágio adicional tanto para um corte horizontal como vertical.

$$F(0, x, y) = \max \{ \langle 0 \rangle \cup \langle v_i : l_i \leq x \ , w_i \leq y, i = 1..n \rangle \}$$

Para G(0, x, y) aparecem os mesmos casos e por isto é idêntico a F(0, x, y).

Considerando o corte no k-ésimo estágio no retângulo (x, y) onde a direção do corte no 1º estágio foi horizontal existem apenas três alternativas para o corte ótimo:

a) É permitido apenas (se possível) o estágio de ajuste em (x, y).

b) Existe pelo menos um corte horizontal, y_1 , no 1º estágio em (x, y). Se um ajuste horizontal é permitido (T = 2 ou T = 4), é necessário considerar apenas valores $y_1 \leq y/2$ (pelo efeito de simetria).

c) Não existe um corte horizontal no 1º estágio em (x, y), mas existe um vertical, x_1, no 2º estágio. Neste caso há um esquema de corte para (x, y) com um corte vertical no 1º estágio e k-1 estágios no esquema de corte final.

Utilizando a propriedade i) definida na seção 4.3.1 e considerando que é possível um estágio de ajuste, o conjunto de possíveis cortes será definido da seguinte forma:

$$\mathbb{L}_1 = \{x : x = \sum_{i=1}^{n} l_i a_i \ , \ 1 \leq x \leq L - k_3 \ , \ a_i \in \mathbb{Z}^+\}.$$

onde:

$$k_3 = \begin{cases} \min (l_i , i = 1..n) & \text{se } T = 3 \text{ ou } T = 4 \\ 1 & \text{caso contrário.} \end{cases}$$

e

$$\mathbb{W}_1 = \{y : y = \sum_{i=1}^{n} w_i a_i \ , \ 1 \leq y \leq W - k_4 \ , \ a_i \in \mathbb{Z}^+\}.$$

onde:

$$k_4 = \begin{cases} \min(w_i , i = 1..n) & \text{se } T = 2 \text{ ou } T = 4 \\ 1 & \text{caso contrário.} \end{cases}$$

Para usar o corte normal na programação dinâmica há necessidade de definir ainda os seguintes conjuntos :

$$p(x) = \max (0, \bar{x} : \bar{x} \leq x, \bar{x} \in \mathbb{L}_1) \ x \leq L$$

e

$$q(x) = \max (0, \bar{y} : \bar{y} \leq y, \bar{y} \in \mathbb{W}_1) \ y \leq W \ .$$

Ou seja, p(x) é o valor mais próximo de x dentro do conjunto de valores normalizados $\mathbb{L}_1$ (p(x) ≤ x , x ∈ $\mathbb{L}_1$ e p(x) = 0, no caso contrário). Para simplificar o algoritmo define-se p(L) = L isso não é estritamente necessário uma vez que pode não existir um conjunto de itens cuja soma dos comprimentos seja exatamente L . Um raciocínio similar aplica-se a q (y) .

Assim, restringindo os valores de x e y àqueles pertencentes aos conjuntos $\mathbb{L}^\circ$ ($\mathbb{L}^\circ = \mathbb{L}_1 \cup \{L\}$) e $\mathbb{W}^\circ$ ($\mathbb{W}^\circ = \mathbb{W}_1 \cup \{W\}$) obtém-se a seguinte equação recursiva:

$$F(k,x,y) = \max \{ F(0, x, y);$$
$$F(k, x, y_1) + F(k, x, q(y-y_1)),\ y_1 \in \mathbb{W}_1 \text{ e } y_1 \leq k_1(y);$$
$$G(k-1, x, y) \}.$$

onde:

$$k \geq 1;\ x \in \mathbb{L}^{\circ} \text{ e } y \in \mathbb{W}^{\circ};$$

e

$$K_1(y) = \begin{cases} y/2 & \text{se } T = 2 \text{ ou } T = 4 \\ y-1 & \text{caso contrário} \end{cases}$$

Com um raciocínio similar ao anterior:

$$G(k,x,y) = \max \{ G(0, x, y);$$
$$G(k, x_1, y) + G(k, p(x-x_1), y),\ x_1 \in \mathbb{L}_1 \text{ e } x_1 \leq k_2(x);$$
$$F(k-1, x, y) \}.$$

onde:

$$k \geq 1;\ x \in \mathbb{L}^{\circ} \text{ e } y \in \mathbb{W}^{\circ};$$

e

$$K_2(x) = \begin{cases} x/2 & \text{se } T = 3 \text{ ou } T = 4 \\ x-1 & \text{caso contrário} \end{cases}$$

O corte ótimo em n-estágios de uma placa (L x W) será dado por F(n, L, W) ou G(n, L, W) se a direção de corte no 1º estágio for especificada, e max { F(n, L, W), G(n, L, W) } no caso contrário. O esquema de corte ótimo pode ser obtido de uma maneira

simples se forem usados apontadores para cada (k, x, y) que indiquem os termos da recursão que levam aos valores F(k, x, y) e G(k, x, y). Uma simples rotina de retrocesso é então usada para recuperar o esquema de corte associado ao valor ótimo.

Estas equações recursivas envolvem $O[n|L_i||W_i|(|L_i|+|W_i|)]$ operações e $O(|L_i||W_i|)$ capacidade de memória. Os resultados computacionais apresentados por Beasley [Be1] sugerem que o uso de três ou mais estágios não conduz a soluções significativamente superiores.

4.3.3 - MÉTODO PARTICIONAR E LIMITAR [H, CW]

O método descrito nesta seção é baseado no trabalho desenvolvido por Herz [H]. O problema abordado aqui é GC com a restrição adicional de um limitante superior, u_i, para cada item da demanda. Este modelo é útil quando se deseja ter um número pequeno de esquemas de corte implementados na produção [Has1].

Os esquemas de corte são gerados desenvolvendo um procedimento de busca em árvore onde no nó raiz as ramificações representam todos os possíveis cortes na placa retângular de dimensões L x W, e o nó ao fim de algum ramo representa os itens produzidos pelo respectivo corte. A cada nó subsequente, um retângulo é selecionado para futuros cortes.

As propriedades i), ii) e iii) são usadas afim de evitar a duplicação de um esquema de corte ou que o algoritmo corte um item que não pertença à demanda.

A propriedade de simetria juntamente com a propriedade de ordenação dos cortes limitam os cortes verticais subsequentes a um

corte feito na posição x = a no retângulo de dimensões (p, q) a serem feitos no maior dos retângulos resultantes, (p-a,q), dentro do intervalo $a \leq x \leq \lfloor (p-a)/2 \rfloor$. Se, $\lfloor (p-a)/2 \rfloor \leq a$ nenhum corte vertical pode ser feito neste retângulo.

Limitar os corte verticais apenas ao valores de x que levam a esquemas de cortes normais, implica em tomar valores de x dentro do seguinte conjunto:

$$\mathbb{L} = [\, x : x = \sum_{i=1}^{n} l_i\, a_i \,,\ 1 \leq x \leq L - 1,\ a_i \in \mathbb{Z}_+ \text{ e } a_i \leq u_i \,].$$

Os resultados descritos acima também são válidos para o corte horizontal, então estes limitam-se aos valores de y pertecentes ao conjunto:

$$\mathbb{W} = [\, y : y = \sum_{i=1}^{n} w_i a_i \,,\ 1 \leq y \leq W - 1 \,,\ a_i \in \mathbb{Z}_+ \text{ e } a_i \leq u_i \,].$$

Além dessas propriedades, para gerar todos os esquemas de corte é necessário incluir, entre todos os possíveis cortes, um corte artificial que deixa o retângulo intacto. Este corte, chamado corte zero, é utilizado para indicar que não são possíveis cortes adicionais.

Para diminuir o número de nós que o algoritmo percorre na árvore são desenvolvidos limitantes superiores para o valor da solução. A cada nó da árvore, existem retângulos que receberam o corte zero, seja $\mathbb{H}_o$ o conjunto formado por estes retângulos, e outros que são candidatos a futuros cortes. Como o esquema de corte é normal tem-se que no esquema de corte final, haverá apenas um item alocado a cada um dos retângulos do conjunto $\mathbb{H}_o$. Um

limitante superior pode ser deduzido em cada nó a partir da alocação ótima de itens nestes retângulos. Para obter este limitante resolve-se o seguinte problema do transporte:

$$\max \quad \sum_{k=1}^{r} \sum_{i=1}^{n} \pi'_{ik} \, x_{ik}$$

$$s.a \quad \sum_{i=1}^{n} x_{ik} \leq 1 \quad k = 1..r$$

$$\sum_{k=1}^{r} x_{ik} \leq u_i \quad i = 1..n$$

$$x_{ik} \geq 0$$

onde:

n = número de itens da demanda

r = número de retângulos pertecentes a H_o

$$\pi'_{ik} = \begin{cases} \pi_i & \text{se } l_i \leq p_k \text{ e } w_i \leq q_k \\ -\infty & \text{caso contrário.} \end{cases}$$

Por causa das restrições deste problema, x_{ik} assume os valores :

$x_{ik} = 0(1)$ caso o item i seja alocado ou não no retângulo (p_k, q_k)

Um retângulo que não recebeu corte zero, está disponível para ser cortado em retângulos menores e portanto vários itens da demanda podem ser alocadas a ele na solução final. Nessa situação é possível obter limitantes resolvendo o problema do corte bidimensional irrestrito através de um procedimento de programação dinâmica. Como a restrição no número máximo de itens da demanda é relaxada, a alocação feita pela programação dinâmica pode ser infactível. Se for factível a alocação obtida é a melhor possível

e portanto nenhuma ramificação é necessária a partir deste nó. No caso contrário, o nó é candidato a futuras ramificações. É bom observar que a solução do problema irrestrito não precisa ser calculada a cada nó, mas uma única tabela de programação dinâmica correspondente à solução do retângulo inicial (L, W) é calculada no inicio do algoritmo. A maior parte do cálculo dos limitantes superiores se refere à solução do problema do transporte e é feita apenas quando o conjunto H_o muda.

Várias regras de partição podem ser usadas em um nó, entre elas:

- selecionar o retângulo de menor dimensões (ou o de maior).
- selecionar o retângulo cuja solução do problema irrestrito fornece o maior valor.

4.3.4 - MÉTODO HEURÍSTICO [W]

Wang [W] trabalha com a maximização da área ocupada no esquema de corte final. Existe uma restrição para o número máximo de cada item da demanda presente em um esquema de corte e o valor de cada item é igual à sua área. Seu algoritmo realiza o processo inverso do corte, isto é, ao invés de enumerar todos os possíveis cortes que podem ser feitos em uma placa, o algoritmo encontra os esquemas de corte acoplando sucessivamente os itens uns aos outros. O número de combinações é reduzido rejeitando os acoplamentos com perda superior a um determinado parâmetro pré-estabelecido.

4.5 - VARIAÇÕES DO PROBLEMA [Sa]

Até aqui foram vistos alguns métodos básicos de resolução para o problema do corte bidimensional. No entanto, na prática, aparecem diversas questões que exigem modificações nos métodos anteriores para que possam ser aplicados.

As questões práticas do problema incluem os seguintes casos:

a) Orientação na placa, granulação ou fibras.

b) Placas que não possuem uma qualidade uniforme ou com regiões defeituosas.

c) Corte de itens não retângulares.

d) Placas de vários tamanhos.

e) Quando se determina os esquemas de corte ótimos, a ordem em que estes esquemas serão cortados é importante sob o ponto de vista que diferentes montagens são necessárias e o custo de cada uma é bastante significativo, podendo até influênciar na determinação dos mesmos. Assim, um objetivo que se coloca é determinar os esquemas de corte minimizando as perdas e o número de montagens.

São diversas as formas de atacar essas questões. A orientação da placa é uma consideração importante a ser feita. No entanto, uma vez determinada a orientação, um método básico pode ser usado.

Uma situação relativamente mais difícil é quando a placa possui defeitos. Hahn [Ha] e Lam [L] propõem diferentes formas de atacar este problema. Hahn usa um método em três estágios semelhante ao método em dois estágios descrito na seção 4.3.1. Lam propõe um método hierárquico usando os conceitos de "quad-cut", "edge-cut", e corte com guilhotina. Cada um destes conceitos desempenhando papeis diferentes na determinação do esquema de

corte ótimo.

Uma forma de simplificar o problema do corte de itens não retângulares, é resolver o problema por um método aproximado que consiste em rodear as formas irregulares por retângulos da forma como segue. Os itens irregulares são tomadas em grupos de dois ou três e ajustados de uma maneira que a área de perda seja minimizada. Esses grupos, chamados da "forma combinada", são então rodeados por um item retângular de área mínima. Essas "formas combinadas" é que são alocadas na placa retângular. O problema então pode ser resolvido por um dos métodos descritos nas seções anteriores.

A variação do problema introduzida pelo item d) acima, isto é, placas de vários tamanhos, pode ser resolvida a partir do método simplex com geração de colunas, basta que a cada iteração se resolva o problema GC para cada um dos tamanhos de placa e se tome o que tiver maior valor total. Se existem muitos tamanhos diferentes, este procedimento vai exigir longo tempo computacional.

Os esquemas de corte obtidos através dos métodos da seção 4.3 são em geral aleatórios. A dificuldade com esquemas de corte aleatórios é que na hora de sequência-los para o corte, trocar de um esquema para outro, pode envolver uma mudança drástica de montagem. Se as placas são de tamanhos diferentes, o problema fica pior ainda. Esse fato trás um aspecto interessante ao problema: minimizar o custo das montagens.

Dois métodos diferentes podem ser usados para resolver essa questão. O primeiro deles, é determinar os esquemas de corte da maneira usual e então sequência-los de forma a minimizar o custo

das montagens. Para obter essa sequência de montagens com custo mínimo, pode ser usado um procedimento igual ao usado para o problema do caxeiro viajante. Cada esquema de corte é considerado como uma cidade, e a matriz de custos é determinada baseada no número de itens novos cortados em esquemas sucessivos. Um esquema de corte artificial é adicionado para a contagem apropriada do número de itens novos. Esse procedimento requer muito esforço computacional.

A outra maneira, é obter os esquemas de corte através de um procedimento que simultaneamente minimize o número de placas e custo de montagem. Alguns procedimentos heurísticos para solucionar o problema com este objetivo são discutidos em Dyson e Gregory [DG].

CAPÍTULO V

5 - EXPERIÊNCIAS COMPUTACIONAIS

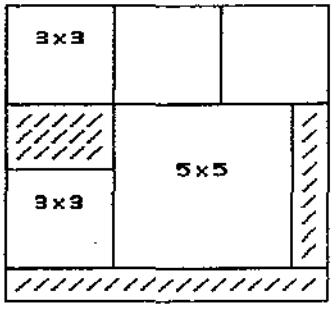
Com o objetivo de completar o estudo do Problema do Corte Bidimensional (cap-IV) foram desenvolvidos, implementados, e comparados três algoritmos baseados no método Simplex Revisado com geração de colunas (seção 1.2.2). O primeiro é o método de geração de colunas em dois estágios (seção 4.3.1). A partir deste método derivou-se mais dois métodos.

São descritos neste capítulo os principais procedimentos usados para geração de colunas em dois estágios; os algoritmos derivados deste; e os resultados dos testes efetuados com os três programas.

5.1 - GERA-COLUNA EM DOIS ESTAGIOS

Como visto na seção 4.3.1, gerar um esquema de corte em dois estágios consiste em resolver n+1 problemas da Mochila Inteiro. O programa implementado resolve os m primeiros problemas da mochila simultaneamente através do método de Programação Dinâmica descrito na seção 2.2.3, e o problema restante através do método de enumeração implícita descrito na seção 2.2.2.

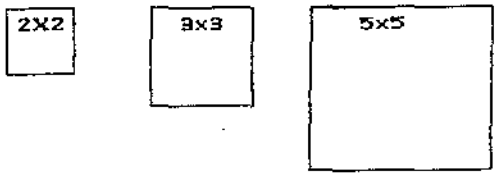
Resolvendo um pequeno exemplo com placa de dimensões (9 x 9) e três itens de dimensões (2 x 2), (3 x 3) e (5 x 5), obtém-se com este método um esquema de corte do tipo:



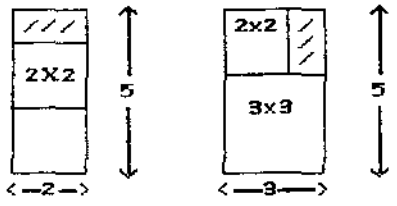
5.2 - NOVO MÉTODO

Analisando o esquema de corte acima, nota-se que na parte hachuriada pode ser alocado um item da demanda (o item 2x2). No entanto, o método em dois estágios contrói faixas de modo que este espaço não é aproveitado, provocando grandes perdas na placa.

A maneira encontrada para reduzir tais perdas é considerar além dos itens originais um conjunto de novos itens que são combinações de itens originais. Para cada largura dos itens originais são geradas todas as combinações de itens originais cuja largura não ultrapasse a largura do item original em questão e cujo comprimento é igual ao do item original de maior comprimento dentre os envolvidos em sua construção. Por exemplo, a partir dos itens:



são construídos os seguintes novos itens:



As faixas são construídas então, considerando este novo conjunto de itens (itens originais + novos itens).

É claro que o número de combinações possíveis pode ser muito grande. Para reduzir este número são usadas as seguintes regras:

i) considera-se apenas os itens originais com valor positivo;

ii) considera-se no máximo n itens do novo conjunto, com comprimentos originais, de maior valor.

Com a mudança do conjunto de itens a ser considerado para a construção das faixas, as n primeiras mochilas do método anterior são resolvidas isoladamente através do método de enumeração implícita descrito na seção 2.2.2 .

5.3 - MODIFICAÇÕES NO NOVO MÉTODO

A idéia exposta acima foi ainda mais explorada e as seguintes modificações foram feitas:

1) Supõe-se L e W valores inteiros pequenos e l_i, $w_i \in \mathbb{Z}_+$.

2) São contruídos novos itens com larguras variando de 1 até W (largura da placa). O número de novos itens é reduzido através das mesmas considerações usadas na seção anterior.

3) São construídas W faixas.

Na implementação deste método, são resolvidas W + 1 problemas da mochila através do método da seção 2.2.2. É claro que devido ao número de problemas da mochila que devem ser resolvidos, que ele tem um tempo computacional dependente da largura da placa considerada para o corte.

5.4. - RESULTADOS

Os programas desenvolvidos foram identificados por:

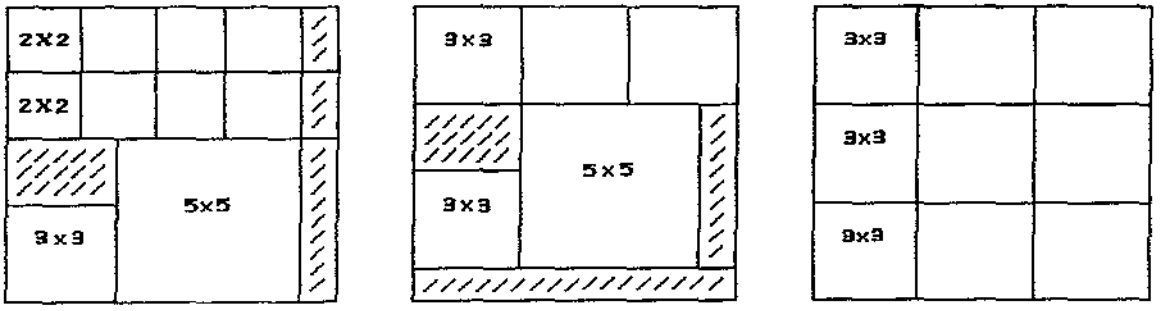
PROG1 - Corte em dois estágios (seção 5.1)

PROG2 - Corte em três estágios (seção 5.2)

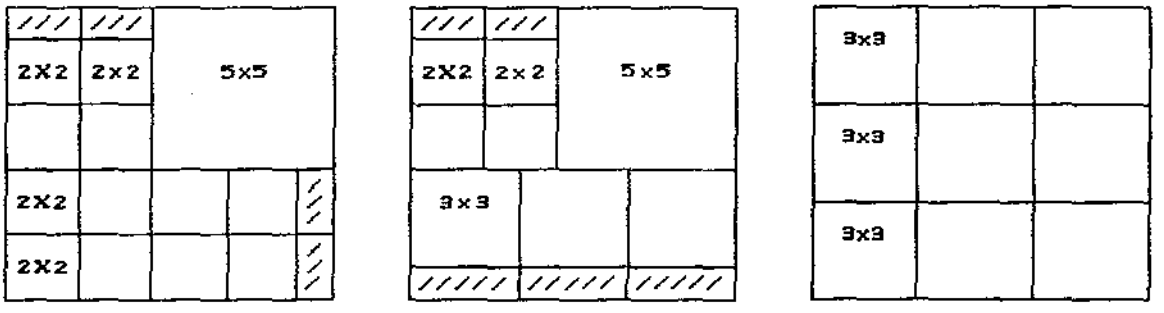
PROG3 - Novo Corte em três estágios (seção 5.3)

Os esquemas de corte obtidos usando PROG1, PROG2 e PROG3 para resolver o problema: (L, W) = (9, 9) ; (l_i , w_i) = ((2, 2), (3, 3), (5, 5)) e d_i = (36, 15, 6) são:

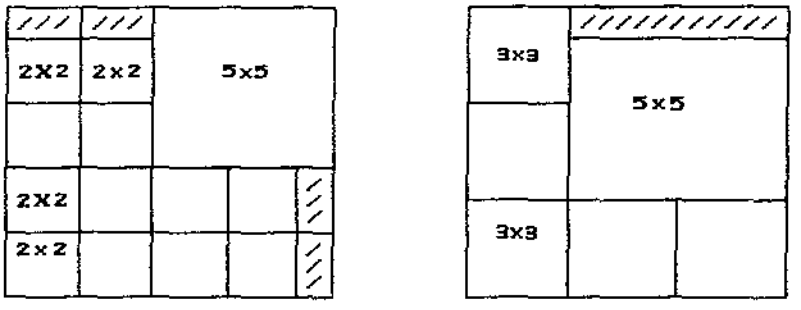
PROG1 - Solução obtida : 6.500



PROG2 - Solução obtida 6.165



PROG3 - Solução obtida 6.000



Foram feitos diversos testes comparativos entre os três programas. Os problemas teste foram gerados considerando placas de tamanhos: (10 x 15), (15 x 10), (50 x 50), (1200 x 1000) e demanda dos itens gerada uniformemente dentro dos seguintes intervalos: [50, 100], [100, 150]. Os problemas teste foram gerados para 10 e 15 itens com dimensões geradas uniformente de acordo com as seguintes relações entre o comprimento e a largura de cada item:

Tipo 1) $l,w \in [i1, i2]$

Tipo 2) $w \in [i1, i2]$, $l = w + i2/10$

Tipo 3) $w \in [i1, i2]$, $l = (U/W)*w$

Tipo 4) $l,w \in [i1, i2]$, $l = \min (l)$, $w = \min (w)$

Tipo 5) $l,w \in [i1, i2]$, $l = \min (l)$, $w = \max (w)$

onde: [i1, i2] = [1, 10], [1,50] e [300, 600].

Os Programas foram escritos em Pascal. Os teste para comparação de tempo de CPU, número de iterações do Simplex, e valor da solução foram realizados em microcomputadores do tipo PC com coprocessador aritmético 8087.

Um teste comparativo entre os três programas analizando o tempo de CPU e o número de iterações do simplex com o número de itens variando de dois em dois dentro do intervalo [4, 20]; itens do tipo 1 gerados no intervalo [1, 10], com demanda no intervalo [50, 100] e placa (10 x 15); apresentou os seguintes resultados para 5 problemas (média ± desvio padrão):

NÚMERO DE ITERAÇÕES

Nº de itens	PROG1	PROG2	PROG3
4	6.40 ± 0.54	6.40 ± 0.54	6.40 ± 0.54
6	10.40 ± 1.10	10.60 ± 1.34	10.80 ± 1.30
8	16.40 ± 3.60	17.20 ± 4.30	17.20 ± 4.60
10	25.00 ± 3.80	24.60 ± 2.87	25.40 ± 4.03
12	26.20 ± 3.90	27.80 ± 2.90	31.80 ± 7.50
14	39.60 ± 11.70	37.60 ± 9.80	40.00 ± 11.90
16	45.00 ± 5.10	53.20 ± 3.50	57.60 ± 11.76
18	45.80 ± 3.10	53.40 ± 8.40	56.60 ± 13.79
20	51.60 ± 6.40	66.80 ± 13.80	63.40 ± 14.36

TEMPO DE CPU ('')

Nº de itens	PROG1	PROG2	PROG3
4	0.63 ± 0.08	1.07 ± 0.14	2.88 ± 1.0
6	1.43 ± 0.19	2.90 ± 0.68	6.32 ± 1.3
8	3.38 ± 0.93	7.59 ± 1.90	13.88 ± 3.5
10	7.51 ± 1.88	18.37 ± 5.48	58.87 ± 70.1
12	10.38 ± 2.06	34.48 ± 8.87	136.05 ± 148.9
14	20.13 ± 6.96	70.64 ± 25.19	200.78 ± 132.2
16	30.80 ± 6.78	135.30 ± 46.23	469.88 ± 551.7
18	35.98 ± 3.89	212.95 ± 117.12	555.45 ± 550.1
20	56.26 ± 14.05	341.78 ± 187.23	683.35 ± 638.9

Em relação ao tempo de CPU, PROG1 é menos sensível a alteração no número de itens do que PROG2 e PROG3, mas só consegue resolver problemas com no máximo 30 itens. PROG2 e PROG3 resolvem problemas com até 50 itens. O tempo de CPU médio de PROG3 tem uma variância muito alta. Em relação ao número de iterações os três métodos apresentam desempenho semelhante.

A segunda parte dos teste foi dividida em dois grupos.

Grupo 1 - Comparação entre PROG1, PROG2 e PROG3

1) Placa (15 x 10), itens do tipo 1 gerados no intervalo [1, 10] e demanda no intervalo [50, 100]. Os resultados médios para 10 problemas foram:

SOLUÇÃO OBTIDA (QUASE ÓTIMA)

	PROG1	PROG2	PROG3
10 itens	258.41 (0 %)	250.90 (- 2.91 %)	250.82 (- 2.94 %)
15 itens	367.23 (0 %)	354.57 (- 3.44 %)	354.15 (- 3.56 %)

TEMPO DE CPU ('')

	PROG1	PROG2	PROG3
10 itens	6.42 (0 %)	17.62 (174.45 %)	17.50 (172.59 %)
15 itens	21.02 (0 %)	83.59 (229.02 %)	54.34 (158.02 %)

N º DE ITERAÇÕES

	PROG1	PROG2	PROG3
10 itens	20.50 (0 %)	24.20 (13.66 %)	24.50 (13.17 %)
5 itens	36.40 (0 %)	43.80 (20.33 %)	42.40 (16.48 %)

Analizando estes resultados observa-se que PROG2 e PROG3 apresentam soluções melhores que PROG1, apesar do número de iterações e do tempo de CPU ser maior. PROG3 apresenta um tempo computacional menor que PROG2.

2) Mudando o tamanho da placa para (10 x 15), PROG2 e PROG3 continuam apresentando soluções melhores que PROG1, apesar do número de iterações e do tempo de CPU ser maior. Neste caso PROG3 apresenta o pior tempo computacional.

3) Para problemas teste com Placa (10 x 15), demanda no intervalo [50, 100] e 10 itens gerados no intervalo [1, 10] com diversas relações entre o comprimento e largura de cada item, os resultados médios para 5 problemas foram:

SOLUÇÃO OBTIDA (QUASE ÓTIMA)

	PROG1	PROG2	PROG3
tipo 4	239.87 (0 %)	239.00 (- 0.36 %)	238.88 (- 0.41 %)
tipo 5	190.89 (0 %)	185.44 (- 2.85 %)	179.39 (- 6.02 %)

TEMPO DE CPU ('')

	PROG1	PROG2	PROG3
tipo 4	7.04 (0 %)	25.24 (258.52 %)	92.57 (1214.91 %)
tipo 5	7.04 (0 %)	23.77 (237.64 %)	67.95 (865.19 %)

Nº DE ITERAÇÕES

	PROG1	PROG2	PROG3
tipo 4	23.20 (0 %)	28.80 (24.14 %)	26.40 (13.79 %)
tipo 5	23.60 (0 %)	28.00 (16.64 %)	30.60 (29.66 %)

Observa-se que PROG3 apresenta para dados do tipo 5 uma solução ótima bem superior que PROG2 e PROG1. Enquanto que o tempo de CPU e o número de iterações é bem inferior.

Grupo2 - Comparação entre PROG1 e PROG2

1) Placa (1200 x 1000), itens do tipo 1 gerados no intervalo [300, 600] e demanda no intervalo [50, 100]. Como neste caso a largura de um item é no máximo 2 vezes maior do que a largura de algum outro item, PROG1 e PROG2 apresentam a mesma solução média. PROG2 apresenta um tempo de CPU maior.

2) Placa (50 x 50), demanda no intervalo [50, 100] e itens gerados em dois intervalos: [1, 10] e [1, 50] com diversas relações entre o comprimento e largura de cada item. Os resultados médios para 5

problemas mostram que PROG2 apresenta uma solução levemente superior (de 0% a 3.32 %) que PROG1, com tempo de CPU e número de iterações maior. PROG1 não consegue resolver problemas teste com itens gerados de acordo com o tipo 3 e dentro do intervalo [1, 10]. Isto se dá porque o número de elementos do conjunto S_1^i (seção 2.2.3) é muito grande e estoura a capacidade de armazenamento da máquina.

5.1 - CONCLUSÃO GERAL

PROG2 e PROG3 apresentam soluções melhores que PROG1 (0% a 6%). Em relação ao tempo de CPU, PROG1 chega à solução do problema cerca de 3 a 5 vezes mais rápido que PROG2 e de 3 a 11 vezes mais rápido que PROG3.

Só existe vantagem do uso de PROG2, ao invés de PROG1, quando a relação entre as larguras dos itens é maior do que 2, ou o número de itens for superior a 30. Caso contrário os dois programas apresentam a mesma solução.

PROG3 sempre consegue soluções melhores que PROG1 e PROG2, mas devido ao tempo de CPU necessário para resolver um problema, seu uso é mais adequado quando a largura da placa for pequena.

BIBLIOGRAFIA

[Be] Beasley, J. E., An exact two dimensional non-guilhotine cutting tree search procedure. **Opns. Res.**, 33(1), 49-64, 1985.

[Be1] ..., Algorithms for unconstrained two-dimensional guilhotine cutting. **J. Opl. Res. Soc.**, 36(4), 297-306, 1985.

[BJ] Bazaraa, M. S. and J. J. Jarvis, **Linear Programming and Network Flows**. Jonh Wiley & Sons, 1979, 565 p..

[C] Christofides, N. et alli, **Combinatorial otimization**. Wiley & Sons, 1979, 425 pgs.

[CW] Christofides, N. and C. Whitlock, An algorithm for two-dimensional cutting problems. **Opns. Res.**, 25(1), 30-44, 1977.

[DG] Dyson, R. G. and A. S. Gregory, The cutting stock problem in the flat glass industry. **Opl. Res. Quart.**, 25(1), 41-53, 1974.

[Dy] Dyckhoff, H., A new linear programming approach to the cutting stock problem. **Opns. Res.**, 6(29), 1092-1104, 1981.

[EC] Eilon, S. and N. Christofides, The loading problem. **Manag. Science**, 17(5), 259-268, 1971.

[F] Ferreira, J.C., "**The cutting stock problem" uma aplicação ao planejamento de produção em fábrica de papel**. Tese de Mestrado, IMECC, UNICAMP, 1988, 55 pgs.

[FH] Fisk, J. and M. S. Hung, A heuristic routine for solving large loading problems. **N. Res. Log. Quaterly**, 26, 643-650, 1979.

[GG1] Gilmore, P. C. and R. E. Gomory, A linear programming approach to the cutting-stock problem. **Opns. Res.**, 9, 849-859, 1961.

[GG2] ..., A linear programming aproach to the cutting stock problem - parte II. **Opns. Res.**, 11, 863-888, 1963.

[GG3] ..., Multistage cutting stock problems of two and more dimensions. **Opns. Res.**, 13, 94-120, 1965.

[H] Herz, J. E., Recursive computational procedure for two-dimensional stock cutting problems. **IBM J. Res. Develop.**, 16, 462-469, 1972.

[Ha] Hahn, S.G., On the optimal cutting of defective Sheets. **Opns Res.**, 16, 1100-1114, 1968.

[Has] - Haesler, R. W., A note on computational modifications to the Gilmore-Gomory cutting stock algorithm. **Opns. Res.**, 28(4), 1001-1005, 1980.

[Has1] - Haesler, R. W., Controlling cutting pattern changes in one-dimensional trim problems. **Opns. Res.**, 23(3), 483-493, 1975.

[HF] Hung, M. S. and J. Fisk, An algorithm for 0-1 multiple knapsack Problems. **N. Res. Log. Quaterly**, 24, 571-579, 1978.

[HS] Horowitz, E. and S. Sahni, Computig partitions with applications to the kanapsack problem. **Journal of ACM**, 21, 277-292, 1974.

[HS1] ..., **Fundamentals of Computer Algorithms**. Computer Science, 1978, 626 pgs.

[IK] Ingargiola, G. P. and J. Korsh, An algorithm for the solution of 0-1 loading problem. **Opns. Res.**, 23, 1110-1119, 1975.

[JDUGG] Johnson, D. S. et alli, Worst case performance bounds for simple one-dimensional packing algorithms. **SIAM J. Computing**, 3(4), 299-325, 1974.

[K] Karmarkar, N., A new polynomial time algorithm for linear programming. **Combinatória**, 4, 373-395, 1984.

[KH] Kennington, J.L. and R. V. Helgason, **Algorithms for network programming**. John Wiley & Sons, 1980, 291 pgs.

[L] Lam, K. P., **A hierarquical method for large scale two-dimensional layout. J. Mech. Tran.**, 105(2), 242-248, 1984.

[La] Lasdon, L. S., **Optimization theory for large systems.** Macmilliam Company, London, 1970.

[MT] Martelo, S. and P. Toth, **Algorithms for knapsack problems.** Notas de aula do curso : Combinatorial Optimization, Núcleo de computação eletrônica, UFRJ, 1985.

[MT1] ..., An upper bound for the zero-One knapsack problem and a branch and bound algorithm. **E. J. Opl. Res.**, 1, 169-175, 1977.

[MT2] ..., A bound and bound algorithm for the zero-One multiple knapsack problem. **D. Appl Math**, 3, 275-288, 1981.

[MT3] ..., Solution of the zero-one multiple knapsack problem. **E. J. Opl. Res.**, 4, 276-283, 1980.

[MT4] ..., Heuristics algorithms for the multiple knapsack problem. **Computing**, 27, 93-112, 1981.

[Mu] Murty, K. G., **Linear and combinatorial programming.** John Wiley & Sons, 1976, 567 pgs.

[S] Salkin, H. M., **Integer programming.** Wesley Wiley & Sons, 1975, 537 pgs.

[Sa] Sarin, S. C., Two-dimensional stock cutting problems and solutions methodologies. **J. Eng. Industry**, 105, 155-160, 1983.

[T] Toth, Paolo, Dynamic programming algorithms for the zero-one knapsack problem. **Computing**, 25, 29-45, 1980.

[W] Wang, P. Y., Two algorithms for constrained two-dimensional cutting stock problems. **Opns. Res.**, 31(3), 573-586, 1983.