versão impressa ISSN 0101-7438 / versão online ISSN 1678-5142

REORDENAMENTO EFICIENTE DAS COLUNAS BÁSICAS NA PROGRAMAÇÃO DE LOTES E CORTES

Glaucia Maria Bressan *
Departamento de Engenharia Elétrica / EESC
Universidade de São Paulo (USP)
São Carlos – SP
glauciab@sel.eesc.usp.br

Aurelio Ribeiro Leite de Oliveira
Inst. de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas (UNICAMP)
Campinas – SP
aurelio@ime.unicamp.br

* *Corresponding author*/autor para quem as correspondências devem ser encaminhadas

Recebido em 06/2003; aceito em 07/2004 após 1 revisão
Received June 2003; accepted July 2004 after one revision

Resumo

Neste trabalho consideramos o problema combinado, que acopla os problemas de dimensionamento de lotes e de corte de estoque, incluindo uma formulação matemática deste problema. Consideramos algumas propriedades da matriz de restrições deste modelo e como construir uma base esparsa para ela, utilizando um reordenamento estático das colunas. Resultados numéricos de uma implementação que realiza trocas de colunas básicas e verifica sua esparsidade, simulando o método simplex são apresentados. Experimentos numéricos também comprovam a robustez desta abordagem. Concluímos que a proposta de construção da base estática esparsa leva a bons resultados computacionais com relação à velocidade e robustez em comparação com abordagens que não consideram a estrutura esparsa da matriz.

Palavras-chave: programação linear; corte de estoque; dimensionamento de lotes.

Abstract

In this work the combined problem is considered, which solves simultaneously the lot sizing and the cutting stock problems. We study some properties of the matrix of constraints and how to factorize the base without losing sparsity in the simplex method context, by a static reordering of the basic columns. Numerical results simulating simplex iterations and verify the sparsity of the factorizations are presented. Numerical experiments had also proven the robustness of this strategy. We conclude that the approach of constructing of the static sparse base reordering leads to very good computational results for both: speed and robustness, in comparison with approaches which do not consider the sparse structure of the matrix of constraints.

Keywords: linear programming; cutting stock; lot sizing.

1. Introdução

As indústrias de manufatura têm sido estimuladas a tornar seus processos mais eficientes devido a aspectos econômicos e avanços computacionais. Isto incentiva o crescimento de modelos de otimização para o controle e planejamento de sistemas produtivos, motivando pesquisas acadêmicas.

O gerenciamento da produção dentro de uma indústria é responsável pelo planejamento e controle da transformação de matérias-primas em produtos finais. O sistema responsável por este gerenciamento denomina-se *Planejamento e Controle da Produção*, que coordena as atividades, desde a aquisição de matérias-primas até a entrega dos produtos finais. Este problema é bem conhecido na literatura e tem motivado pesquisas acadêmicas (Nonas *et al.*, 2000).

O objetivo deste trabalho consiste na resolução dos sistemas lineares provenientes do método simplex, explorando a estrutura matricial inerente ao problema de programação de lotes e cortes. Este é um modelo cuja matriz de restrições possui uma estrutura bloco angular que é acoplada por restrições adicionais. Os blocos matriciais têm alto grau de esparsidade e pouco acoplamento, indicando que esta abordagem deve produzir bons resultados em termos de eficiência computacional.

A solução eficiente de sistemas lineares de grande porte é de fundamental importância na resolução de problemas de otimização. A solução pode ser obtida através de métodos genéricos, via decomposição *LU* das matrizes que formam as bases no método simplex, ou através da exploração da estrutura da classe de problemas a ser resolvida (Luenberger, 1984).

Este artigo está organizado do seguinte modo, na Seção 2 são introduzidos o problema de corte de estoque e o método de geração de colunas. Na Seção 3 é definido o problema combinado e formulado seu modelo matemático, de onde extraímos a matriz de restrições. Na Seção 4 é exibida uma base inicial desta matriz com suas propriedades mais relevantes. A Seção 5 contém os experimentos numéricos realizados em MATLAB e a decomposição LU proposta, que considera a esparsidade da matriz.

2. Problema de Corte de Estoque e Geração de Colunas

Suponha que várias barras estejam disponíveis para serem cortadas na produção dos diversos itens. Temos então que escolher quantas barras devem ser cortadas e como cortá-las. Os objetos em estoque de mesmo tipo estão disponíveis em grande quantidade e podem ser de um único ou de vários tipos, havendo ou não limitação de estoque. A solução deste problema terá muitas peças em estoque igualmente cortadas para a produção dos diferentes tipos de itens.

Chamamos de *padrão de corte* a maneira que um objeto em estoque é cortado para produzir peças menores.

As possíveis combinações entre as peças que devem ser produzidas são muitas. Desta forma, teremos um número muito elevado de padrões de corte. Geralmente, este problema é formulado como um problema de otimização inteira, que é muito difícil de ser resolvido computacionalmente de forma exata. Para lidar com essas dificuldades, Gilmore & Gomory (1961) relaxaram a condição de integralidade e propuseram um método eficiente de geração de colunas para resolver o problema de otimização linear contínua.

Cada padrão de corte corresponde a uma coluna. Assim, o método simplex com geração de colunas consiste em, a cada iteração, substituir uma coluna (um padrão de corte) básica por uma nova coluna (um novo padrão de corte) que melhora a solução corrente.

Determinar a melhor coluna para entrar na base consiste em resolver um subproblema conhecido na literatura como *Problema da Mochila*, no qual um padrão de corte deve satisfazer:

$$l_1\alpha_1 + l_2\alpha_2 + \ldots + l_P\alpha_P \leq L,\ \alpha_p \in Z_+^* \text{ com } p = 1, \ldots, P,$$

onde P é o número de peças, L é o comprimento total do objeto, l_p é o comprimento da peça tipo p e α_p é a quantidade de peças no padrão de corte (α_1, α_2, …, α_P). Ou seja, a soma total do comprimento das peças que estão contidas no padrão de corte deve ser menor ou igual ao comprimento do objeto a ser cortado.

Portanto, no problema de corte de estoque unidimensional, a cada iteração do método simplex será necessário resolver o problema da mochila para determinar a coluna com o menor custo relativo (Chvátal, 1983; Marques, 2000; Martello & Toth, 1989).

3. Problema Combinado

O processo de programar a produção pode ser dividido em três etapas: a primeira define uma carteira de pedidos para um horizonte de planejamento finito, especificando as quantidades dos produtos finais demandados e suas respectivas datas de entrega. A segunda etapa converte a demanda de produtos finais em demanda de peças. Deste modo, um tipo de peça pode ser utilizado por vários produtos finais diferentes. Finalmente, a terceira etapa decide a quantidade de produtos finais que devem ser produzidos em cada período, minimizando os custos e as perdas ocorridas no processo de corte das placas em peças.

Freqüentemente existem perdas de material no corte de peças. Esta perda tende a ser relativamente menor conforme a demanda de peças aumenta, devido a um melhor rearranjo dos padrões de corte nas placas. Portanto surge uma pressão econômica para fabricar alguns produtos antecipadamente de modo a minimizar as perdas. Por outro lado, os custos de estoque exercem pressão oposta no sentido de retardar a produção. Baseado nesta decisão de antecipar ou não a produção de certos produtos finais surge o *Problema Combinado*, que acopla dois importantes problemas de otimização, o *dimensionamento de lotes* e o *corte de estoque* (Gramani, 2001).

3.1 Problema de Dimensionamento de Lotes e Problema de Corte de Estoque

Vejamos os problemas de corte de estoque e de dimensionamento de lotes descrevendo separadamente cada um deles.

Problema de Dimensionamento de Lotes (PDL): Consiste em planejar a quantidade dos itens a ser produzida em vários estágios, em cada período ao longo de um horizonte de tempo finito, de modo a atender a demanda e otimizar uma função objetivo, como minimizar os custos de produção e de estocagem. Um PDL pode ser classificado como *monoestágio*, onde os itens são produzidos independentemente, e *multiestágio*, quando as produções dos itens são dependentes. Para resolvermos este problema, podemos decompô-lo em M subproblemas com apenas um produto final, que podem ser resolvidos, por exemplo, por programação dinâmica (Wagner & Whitin, 1958).

Problema de Corte de Estoque (PCE): Consiste em cortar um conjunto de objetos disponíveis em estoque em peças menores otimizando uma certa função objetivo, por exemplo, minimizar a perda total ou o número de objetos a serem cortados. Algumas regras são necessárias para definir o processo de corte, como cortes do tipo *guilhotinado* (por exemplo, onde cada corte sobre uma placa retangular produz duas novas peças retangulares), limitação de peças (cortes restritos ou irrestritos), número de estágios (é dito ser 2-estágios quando apenas uma mudança no sentido dos cortes guilhotinados é permitida: horizontal/ vertical ou vertical/horizontal). Além disto, o problema será *bidimensional* quando duas dimensões são relevantes para o corte. Para resolver este problema, podemos aplicar o método simplex em conjunto com a técnica de geração de colunas (Gilmore & Gomory, 1965).

3.2 Resolução Prática do Problema Combinado

Em situações reais, a maioria das indústrias aborda esses dois problemas de forma separada, devido à alta complexidade do Problema Combinado. Entretanto, tratá-los de forma separada pode elevar os custos globais, principalmente se uma parcela significativa do custo do produto final é formada pelo material a ser cortado. Para resolvermos os problemas de Corte de Estoque e de Dimensionamento de Lotes separadamente, podemos aplicar uma *Heurística da Decomposição* (Gramani, 2001), que consiste em:

1. Resolver o PDL, em que realizamos o planejamento da produção, decidindo a quantidade de cada tipo de produto final a ser produzido em cada período do horizonte de planejamento, minimizando os custos de produção e estoque;
2. Para cada período, resolvemos um problema de corte de estoque com restrições de capacidade, e assim determinamos a quantidade de placas cortadas conforme um certo padrão;
3. A função objetivo do problema combinado é dada pela soma das funções objetivos dos problemas PDL e PCE.

Apesar das dificuldades do modelo combinado quanto à integralidade das variáveis que representam a quantidade de placas cortadas num certo padrão e a grande quantidade de padrões de corte que podem ser gerados, esta abordagem fornece um ganho significativo nos custos globais, incentivando estudos nesta linha.

3.3 Formulação Matemática do Problema Combinado

O Problema Combinado consiste em decidir a quantidade de produtos finais a serem produzidos em cada período do horizonte de planejamento tal que minimize os custos da produção e estocagem (PDL) e a quantidade de placas a serem cortadas para compor produtos finais (PCE). Uma consideração importante nos modelos reais é a disponibilidade da serra em cada período. Temos de assegurar que o tempo gasto para cortar as placas não excede o tempo disponível.

3.3.1 O Modelo

Uma forma de resolver o Modelo Combinado desconsidera a ocorrência de custos de preparação e relaxa a integralidade das variáveis que representam a quantidade de placas cortadas, o que pressupõe grandes quantidades de demanda. O modelo pode ser aplicado na

indústria de móveis, onde placas de madeira devem ser cortadas na produção de itens. Consideramos que haja apenas um tipo de placa em estoque, $L \times W$, suficiente para atender a demanda. Definimos então o modelo da seguinte forma (Gramani, 2001; Nonas, 2000):

Conjuntos:
$t = 1, ..., T$ número de períodos.
$p = 1, ..., P$ número de diferentes tipos de peças a serem cortadas.
$j = 1, ..., N$ número de diferentes padrões de corte.
$i = 1, ... , M$ número de diferentes produtos finais demandados.

Parâmetros:
c_{it} : custo de produção do produto final i no período t.
h_{pt} : custo de estocagem da peça tipo p no período t.
hf_{it} : custo de estocagem do produto final i no período t.
d_{it} : demanda do produto final i no período t.
r_{pi} : número de peças tipo p necessárias para formar um produto i.
v_j : tempo gasto para cortar uma placa no padrão de corte j.
a_{pj} : número de peças tipo p no padrão j.
u_t : tempo máximo de operação da serra.
cp : custo da placa a ser cortada.

Variáveis:
x_{it} : quantidade do produto final i produzido no período t.
e_{pt} : quantidade da peça tipo p em estoque no fim do período t.
f_{it} : quantidade do produto final i em estoque no fim do período t.
y_{jt} : quantidade de placas cortadas usando o padrão j no período t.

$$min \sum_{i=1}^{M} \sum_{t=1}^{T} (c_{it} . x_{it} + hf_{it} . f_{it}) + \sum_{j=1}^{N} \sum_{t=1}^{T} cp . y_{jt} + \sum_{p=1}^{P} \sum_{t=1}^{T} h_{pt} . e_{pt} \qquad (1)$$

$$s.a: \quad x_{it} + f_{i,t-1} - f_{it} = d_{it} \qquad \forall t = 1,\ldots,T, \quad \forall i = 1,\ldots,M \qquad (2)$$

$$\sum_{j=1}^{N} a_{pj} . y_{jt} + ep_{p,t-1} - ep_{pt} = \sum_{i=1}^{M} r_{pi} . x_{it} \qquad \forall t = 1 \ldots T \qquad (3)$$

$$\sum_{j=1}^{N} v_j . y_{jt} \leq u_t \qquad \forall t = 1,\ldots,T, \quad \forall j = 1,...,N \qquad (4)$$

$$x_{it} , f_{it} , y_{jt} , e_{pt} \geq 0 \qquad \forall t = 1,\ldots,T \qquad (5)$$

A função objetivo (1) minimiza o custo dos produtos finais e o custo do processo de corte. As restrições (2) se referem às equações de balanço de estoque com relação aos produtos finais, o que garante que a demanda de itens de cada período será atendida. As restrições (3) se referem às equações de balanço de estoque com relação às peças, o que asseguram que a demanda de peças será satisfeita. Estas restrições são as que acoplam os problemas de dimensionamento de lotes e de corte de estoque, pois ambas incluem as variáveis x_{it}, que definem o tamanho dos lotes e y_{jt}, que definem a quantidade de placas cortadas num certo padrão de corte. As restrições (4) se referem à capacidade da serra, o que garante que o tempo gasto no processo de corte das placas nos diversos padrões de corte não ultrapassa a capacidade disponível da serra, ou seja, seu tempo máximo de operação.

Este é um modelo linear contínuo, uma vez que a integralidade das variáveis não é considerada. Porém permanece a dificuldade referente à grande quantidade de padrões de corte que podem ser gerados. Neste trabalho, caso a integralidade fosse considerada, esta discussão continuaria válida, pois estamos interessados somente na decomposição *LU* da base do método simplex. Por simplicidade de notação reescrevemos o modelo da seguinte forma:

$$min \sum_{t=1}^{T} (c_t . x_t + hf_t . f_t) + \sum_{t=1}^{T} cp . y_t + \sum_{t=1}^{T} h_t . e_t \qquad (1)$$

s.a: $x_t + f_{t-1} - f_t = d_t \qquad \forall t = 1,\ldots,T \qquad (2)$

$$-R . x_t + A . y_t + e_{t-1} - e_t = 0 \qquad \forall t = 1\ldots T \qquad (3)$$

$$v^T . y_t \leq u_t \qquad \forall t = 1,\ldots,T \qquad (4)$$

$x_t, f_t, y_t, e_t \geq 0 \qquad \forall t = 1,\ldots,T$, onde e_0, f_0 são conhecidos, (5)

$c_t = (c_{1t}, c_{2t}, \ldots, c_{Mt})$; e os demais parâmetros e variáveis hf_t, f_t, h_t, e_t, d_t, x_t e y_t são definidos de forma similar, $v^T = (v_1 \;\; v_2 \;\; \ldots \;\; v_N)$,

$$R = \begin{bmatrix} r_{11} & r_{12} & \ldots & r_{1M} \\ r_{21} & r_{22} & \ldots & r_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ r_{P1} & r_{P2} & \ldots & r_{PM} \end{bmatrix}$$

e A é uma matriz $P \times N$ tal que cada coluna corresponde a um padrão de corte.

3.3.2 A Matriz de Restrições para o Modelo Combinado

Reordenamos a matriz de restrições de modo que adquira formato bloco diagonal, invertendo as posições das variáveis f_t e x_t evitando assim, o preenchimento da matriz. Além disso, acrescentamos as *variáveis de folga* $s_t = u_t - v^T y_t$ para $t = 1\ldots T$, relacionadas às restrições de tempo máximo de operação da serra.

Pelo reordenamento das colunas, é possível obter uma decomposição esparsa para qualquer base, sem nenhum esforço computacional de modo a obter a ordem das colunas, pois o reordenamento da matriz de restrições é estático, e o das colunas da base obedece esta ordem. Desta forma, o reordenamento não tem custo de inicialização, nem de atualização, pois é válido para todas as iterações do método simplex.

As colunas da matriz de padrões de corte não são reordenadas entre si *a priori*, mesmo porque elas são geradas durante a execução do método, e portanto, não são conhecidas. Mas se torna fácil a tarefa de encontrar a posição de tais colunas quando entrarem na base com respeito às outras colunas de padrão no mesmo estágio de tempo. O critério mais prático seria colocá-las em ordem crescente do número de elementos não nulos.

A matriz R, que representa a demanda de peças, pode conter esparsidade, pois, como cada uma de suas colunas representa um produto final, este pode não ser constituído de algumas peças que serão utilizadas para compor um outro produto. Desta forma, as variáveis x_{it}, que

definem o tamanho do lote, podem ser reordenadas de acordo com a matriz *R*, reordenada também em ordem crescente do número de elementos não nulos. Isto equivale a numerar os itens finais em uma ordem conveniente.

Tabela 1 – Matriz de Restrições para o Problema Combinado.

f_1	f_2	...	f_T	x_1	...	x_T	y_1	...	y_T	e_1	e_2	...	e_T	s_1	...	s_T	d_t
-I	0	...	0	I	...	0	0		0	0	...		0	0	...	0	d_1-I_0
I	-I	...	0	0	...	0	0		0	0	...		0	0	...	0	d_2
	...						0		0	0	...		0		...		...
0	0	...	0	0	...	0	0		0	0	...		0	0	...	0	d_{t-1}
0	0	...	0	0	...	0	0		0	0	...		0	0	...	0	d_t
		...		...							...				...		...
0	0	...	-I	0	...	I	0		0	0	...		0	0	...	0	d_T
0		...	0	-R	...	0	A	...	0	-I	0	...	0	0	...	0	0
0		...	0	0	...	0	0	...	0	I	-I	...	0	0	...	0	0
0		...	0		...			...				...			...		0
0		...	0	0	...	-R	0	...	A	0	0	...	-I	0	...	0	0
0		...	0	0	...	0	v^T	...	0	0	0	...	0	1	...	0	u_1
		...			...			...				...		...	...		...
0		...	0	0	...	0	0	...	v^T	0	0	...	0	0	...	1	u_T

É vantajoso que as colunas básicas respeitem este ordenamento, pois a matriz de restrições possuirá formato bloco diagonal, de modo que causará o menor preenchimento da base ao realizarmos sua decomposição LU.

4. Base Inicial para a Matriz de Restrições

Uma base inicial pode ser tomada considerando vazios os estoques de peças e de produtos finais e considerando a matriz dos padrões de corte diagonal, através da escolha dos padrões homogêneos. Assim, obtemos a base inicial mostrada na Figura 1:

x_1	x_2	...	x_T	y_1	y_2	...	y_T	s
$I_{M\times M}$	0	...	0	0	0	...	0	0
0	$I_{M\times M}$	...	0	0	0	...	0	0
		...				...		
0	0	...	$I_{M\times M}$	0	0	...	0	0
$-R_{P\times M}$	0	...	0	$D_{P\times P}$	0	...	0	0
0	$-R_{P\times M}$	...	0	0	$D_{P\times P}$	...	0	0
		...				...		
0	0	...	$-R_{P\times M}$	0	0	...	$D_{P\times P}$	0
0	0	...	0	$v^T_{B_{1\times P}}$	0	...	0	
0	0	...	0	0	$v^T_{B_{1\times P}}$	...	0	I
0	0	...	0	0	0	...	$v^T_{B_{1\times P}}$	

Figura 1 – Base Inicial para a Matriz de Restrições.

4.1 Propriedades da Matriz de Restrições

A seguir, podemos apontar algumas propriedades relevantes da matriz de restrições especialmente quanto à sua decomposição e, observando a forma geral da matriz de restrições, podemos apontar algumas propriedades para a construção da base.

1. Cada y_i pode ter no máximo $P+1$ colunas na base. Isto nos sugere o seguinte teorema:

   Teorema 4.1. *Um subconjunto de colunas das matrizes*

   $$\begin{pmatrix} A \\ v^T \end{pmatrix}$$

   tem no máximo T(P+1) colunas na base.

   Prova: Cada matriz A possui dimensão $P \times N$ e cada vetor v^T possui dimensão $1 \times N$. Então cada y_i possui $P+1$ linhas e N colunas, e todo o conjunto possui $T(P+1)$ linhas e NT colunas. Assim, se cada y_i tiver $P+1$ colunas, este bloco será quadrado. Caso tenha menos de $P+1$ colunas, podemos tomar o bloco de folga como pivô, e o conjunto de matrizes acima é retangular em pé, ou seja, possui mais linhas que colunas. Se cada y_i tiver mais que $P+1$ colunas, a base seria singular. Portanto, cada y_i tem no máximo $P+1$ colunas na base. Logo, todo o conjunto acima deve ter no máximo $T(P+1)$ colunas na base.

2. As matrizes formadas pelas colunas e_p são sempre triangulares inferiores pela restrição $-R.x_t + A.y_t + e_{t-1} - e_t = 0$, e no reordenamento de colunas proposto nunca são decompostas, por conterem apenas dois blocos matrizes identidades.

3. O *bloco folga* também não é afetado pela decomposição, pois possui apenas colunas unitárias e o elemento não nulo de cada coluna básica será tomado como pivô pela decomposição.

4. Conseqüentemente, o esforço computacional da decomposição LU para o reordenamento proposto existe de fato apenas em $\begin{pmatrix} A \\ v^T \end{pmatrix}$ e em R.

5. Entre f_t e x_t deve haver pelo menos M colunas na base. Caso contrário, teríamos uma linha de zeros na matriz, e então ela seria linearmente dependente. Esta afirmação sugere o seguinte teorema:

   Teorema 4.2 *Se $f_t(i)$ não está na base então $x_t(i)$ está na base e podemos tomar $x_t(i)$ como próximo pivô, onde i é a coluna desta matriz, onde existem estoque e produção simultaneamente, caso $x_t(i)$ pertença à base, para $i = 1, \ldots, j$, sendo j o primeiro estágio de tempo. Para os estágios de tempo seguintes, se $x_t(k)$ não está na base, podemos tomar o estoque deste mesmo produto no tempo anterior $f_t(k-1)$, para $k = j+1, \ldots, M$*

   Prova: Se nenhuma das colunas $f_t(i)$ e $x_t(i)$ estiver na base, teremos uma linha de zeros na matriz. Se ambas estiverem, tomamos o primeiro elemento de $-R$ como pivô. E se apenas a coluna $x_t(i)$ estiver na base, e não a coluna $f_t(i)$, o próximo pivô será $x_t(i)$. Quanto aos estágios de tempo seguintes, se $x_t(k)$ não está na base, podemos tomar a

coluna de estoque deste produto final correspondente ao período anterior, pois esta conterá uma matriz identidade, devido à restrição $x_t + f_{t-1} - f_t = d_t$, para $t = 1, \ldots, T$, o que evitará que a base seja singular.

6. Como conseqüência do teorema acima, podemos escrever o seguinte corolário:

   Corolário 4.2.1. A base deve conter pelo menos TM colunas entre os blocos item e produção.

4.2 Identificação do Tipo das Colunas

Dadas a coluna que sai da base e a que entra, podemos identificar a qual bloco pertencem estas colunas, utilizando a posição do primeiro e do último elemento não nulo, pelo algoritmo descrito abaixo. Portanto, são necessárias poucas alterações em uma implementação pré-existente que não considera a estrutura particular da matriz de restrições.

Dada uma coluna c, obtemos a posição do seu primeiro e último elemento não nulo K_1 e K_2:

se $K_1 \leq TM$
  então se $K_2 > TM$
    então $c \in$ produção
    senão $c \in$ estoque de item
senão se $K_1 > T(P+M)$
  então $c \in$ folga
  senão se $K_2 > T(P+M)$
    então $c \in$ padrão de corte
    senão $c \in$ estoque de peça.

Pode-se incluir a atualização da decomposição LU proposta em uma implementação, bastando que as colunas que entram e saem da base sejam fornecidas.

5. Experimentos Numéricos

O objetivo destes experimentos é verificar a viabilidade das idéias apresentadas neste trabalho. No experimento numérico descrito a seguir consideramos: 6 períodos de tempo, 2 produtos finais, 5 peças diferentes e 60 padrões de corte.

Com esses parâmetros, a matriz de restrições possui dimensão 48 x 420, sendo 360 colunas correspondentes a padrões de corte. Para desenvolvermos o experimento, definimos as matrizes B_1, B_2 e B_3. Inicialmente todas elas correspondem à base inicial. Vamos simular iterações do método simplex escolhendo aleatoriamente uma coluna para sair da base e uma para entrar na base. A cada iteração estas três matrizes contêm as mesmas colunas básicas, embora em ordem diferente. As colunas em B_1 respeitam o ordenamento descrito na Tabela 1. Na matriz B_2 sempre colocamos a coluna que entra na base na posição da coluna que sai e na matriz B_3 a coluna que entra na base ocupa a última posição da matriz.

Os experimentos foram realizados no MATLAB 5.3, em um microcomputador Pentium 4 com processador INTEL 1.8GHz e 512MB de memória RAM.

5.1 Critério para Trocas de Colunas

Realizamos trocas das colunas básicas por colunas não básicas, testando a singularidade e verificando a esparsidade. Sorteamos uma coluna que sairá da base e uma que será inserida. Se a nova base for não singular, atualizamos as posições das colunas básicas em B_1, B_2 e B_3, da forma descrita anteriormente; caso contrário, sorteamos outra coluna para entrar na base até que encontremos uma matriz não singular. Efetuamos a seguir a decomposição LU de B_1, B_2 e B_3, e calculamos o número de elementos não nulos dessas decomposições.

Esperamos que B_1 seja mais esparsa que B_2 e B_3, porque ela é ordenada de forma que sua decomposição cause menor preenchimento na matriz.

5.2 Resultados Observados

Apresentamos nas Tabelas 2, 3 e 4 os resultados obtidos pelos experimentos numéricos. Os valores *min*, *max* e *média* nas Tabelas 2 e 3 representam, respectivamente, o mínimo, o máximo e a média do número de elementos não nulos das matrizes L e U ($nnz\,(L+U)$) da decomposição das bases correspondentes. A singularidade da matriz é verificada pela função *rcond( )* do MATLAB, e o cálculo da decomposição *LU* foi realizado usando o comando interno *lu( )*, que reordena as linhas para a seleção do pivô. Antes da primeira troca de colunas, as matrizes B_1, B_2 e B_3 são iguais. Os resultados apresentados consideram diversos números de iterações.

Os resultados obtidos para as 500 primeiras iterações foram os seguintes:

Tabela 2 – Número de elementos não nulos da decomposição *LU* das bases para as 500 primeiras iterações.

nnz(L+U)	B_1	B_2	B_3
min	310	301	389
max	393	556	389
média	351,4	377,6	341,6

Os resultados obtidos para as 10000 primeiras iterações foram os seguintes:

Tabela 3 – Número de elementos não nulos da decomposição *LU* das bases para as 10000 primeiras iterações.

nnz(L+U)	B_1	B_2	B_3
min	293	293	140
max	416	648	491
média	349,5	382,6	354,1

Finalmente a Tabela 4 compara os números de elementos não nulos da decomposição de B1 para as 500 primeiras iterações utilizando diversos valores de *threshold* para pivoteamento.

Tabela 4 – Número de elementos não nulos da decomposição LU de B1 para 500 iterações.

Nnz(L+U)	B_1;1	B_1; 0,1	B_1; 0,01
min	304	310	257
max	429	393	415
média	388,2	351,4	356,9

Para este nosso exemplo, relativamente pequeno, o valor de *threshold* (B_1; 0,1) apresentou melhores resultados que os demais, por ter se mostrado mais esparso.

5.3 Decomposição *LU* com Troca de Colunas

Implementamos em MATLAB uma decomposição LU que considera a estrutura esparsa da matriz de restrições, e comparamos nossos resultados com o cálculo da decomposição *LU* completa, realizada através do comando interno do MATLAB *lu( )*, que não explora a esparsidade da matriz.

Como a base B_1 havia se mostrado mais esparsa que as bases B_2 e B_3 nos experimentos anteriores, comparamos então o número de operações necessárias para decompormos esta base, através da contagem de *flops*, ou seja, *contagem de operações de ponto flutuante*, das formas de decomposição. A comparação foi feita usando três formas de decomposição diferentes.

A primeira é a decomposição *LU* completa, que não explora a estrutura matricial, ou seja, a esparsidade da matriz, que chamamos de *lu(B)*. Na segunda forma de decomposição, a base da matriz de restrições é redecomposta a partir da posição da primeira coluna que entrar ou sair da base. Para tanto, tomamos a posição na base da coluna que sai (s) e da que entra (e), e escolhemos a menor delas (j). Na Tabela 5 , esta decomposição está indicada como *min(e, s)*. A terceira e última forma, que chamamos *dec.esparsa(B)*, redecompõe apenas as colunas necessárias, pois a saída de uma coluna pode não alterar as demais colunas pela sua estrutura esparsa.

Nesta nova decomposição, se $s < e$, estamos interessados em saber quantas colunas a partir de s+1 não são afetadas pela saída desta coluna s. A decomposição é feita apenas nas colunas afetadas, isso porque a saída de uma coluna pode não afetar colunas posicionadas depois dela pela sua estrutura *esparsa*. Ou seja, as posições dos elementos não nulos da coluna s podem não coincidir com os elementos não nulos de alguma coluna seguinte, e então, essa coluna não sofrerá qualquer alteração de s, e então não será decomposta.

As operações que foram causadas pela coluna que saiu são *desfeitas*. Para tanto, efetuamos as operações na ordem inversa da decomposição de k, sendo $k = s+1 \ldots e-1$, sempre considerando a esparsidade. Também decompomos e, a coluna que entra na base, e contamos o número de *flops* desta operação.

Agora, se $e < s$, isto é, se a coluna a entrar na base for inserida em alguma posição anterior à posição da coluna a sair da base, se necessário, decompomos novamente k, neste caso, para $k = e+1 \ldots s-1$, e calculamos o número de operações.

Finalmente, resta-nos decompor as últimas colunas da base, ou seja, devemos decompor a partir de *max(e, s)* até a última coluna da base, m. Então, se necessário, fazemos a decomposição da coluna k, neste caso, para $k = max(e,s) \ldots m$, e calculamos o número de *flops* desta atualização.

5.3.1 Comparação dos Resultados

A Tabela 5 abaixo exibe a comparação do número de *flops* (contagem de operações de ponto flutuante) entre a decomposição proposta neste trabalho (dec.esparsa(B)), a decomposição a partir da primeira coluna afetada por *j*, e a decomposição completa, utilizando o comando do MATLAB.

Os resultados a seguir mostram o mínimo, a média e o máximo de *flops* entre as duas decomposições, para 500 iterações.

Tabela 5 – Contagem de operações entre as decomposições de B_1.

flops	*lu(B)*	*min(e, s)*	*dec.esparsa(B)*
min	1241	37	7
max	1477	391	71
média	1352,9	185,1	33,7

Podemos concluir que a decomposição *LU* proposta neste trabalho reduz o número de operações em aproximadamente 97% se comparada à versão que não explora a decomposição da base anterior, podendo realizá-la de maneira mais rápida e eficiente. Mesmo explorando parcialmente a esparsidade, como acontece na decomposição indicada por *min(e, s)*, onde a base é redecomposta a partir da posição da primeira coluna que entrar ou sair da base, já temos um ganho significativo de em média 86% se comparada à decomposição completa. Como vimos, podemos obter um resultado ainda melhor quando exploramos totalmente a esparsidade, como é mostrado na Tabela 5 com a *dec.esparsa(B)*.

Por estarmos utilizando o MATLAB nas implementações, é mais relevante ser exibida a contagem de *flops* do que o tempo de processamento das operações, que reflete melhor quais seriam os tempos relativos em outra linguagem de programação como C ou Fortran.

5.3.2 Estimativa do Erro da Atualização

Testes computacionais adicionais foram realizados com o objetivo de verificar a robustez do método proposto. Para isso, todas as operações da decomposição foram "desfeitas" em todas as iterações do método simplex. Ao efetuarmos as operações na ordem inversa da decomposição *LU* obtemos uma matriz que após a troca das colunas que entra e sai, simulará a atualização da decomposição da base ao ser decomposta desde a primeira coluna. Optamos por desfazer toda a decomposição, com o objetivo de ilustrar a robustez do método de atualização proposto, pois estamos simulando o pior caso em termos do número de operações realizadas, após cada iteração.

Como medida do erro introduzido por estas operações, calculamos a norma entre a matriz, obtida a partir da função que desfaz as operações da decomposição, e a base reordenada, obtida diretamente da matriz de restrições, para várias iterações do simplex.

Outra maneira de recuperar a matriz base inicial é efetuar a operação L*U, onde *L* é a matriz triangular inferior com elementos da base, e *U* a matriz triangular superior.

Os erros máximo e médio obtidos para 100, 1000, 2000 e 10000 iterações são apresentados na Tabela 6. As duas primeiras linhas exibem o erro acumulado para a operação que desfaz toda a decomposição, e as duas últimas linhas exibem do erro da recuperação da base através da multiplicação de *L* por *U*.

Tabela 6 – Estimativa do erro de aproximação da atualização da base.

Erro	100	1000	2000	10000
max (desfaz)	6,8 e-13	1,3 e-12	1,8 e-12	4,0 e-12
média (desfaz)	1,6 e-13	6,1 e-13	5,8 e-13	7,0 e-13
max (L*U)	1,1398 e-12	2,2397 e-12	2,8235 e-12	3,6005 e-12
media (L*U)	4,2917 e-13	1,1003 e-12	9,9500 e-13	1,1265 e-12

Podemos verificar que o método de atualização da base se mostrou bastante robusto, pois mesmo realizando 10000 iterações o erro absoluto acumulado na matriz da base no pior caso é da ordem de 10^{-12} em relação às colunas originais. Estes resultados são ainda mais significativos ao verificarmos que o erro ao recuperarmos as matrizes utilizando o produto *L*U* é na média ligeiramente superior que a operação de desfazer as decomposições. Isto significa que não é necessário calcular periodicamente uma decomposição da base, para esta aplicação, como em outros métodos de atualização, pois a abordagem proposta se mostrou extremamente robusta para este experimento.

6. Conclusões

Construímos uma base esparsa para esta matriz de restrições através de um reordenamento estático das colunas básicas, o que nos proporcionou uma decomposição esparsa para qualquer base, e por ser estático, não requer esforço computacional para obter a ordem das colunas. Pelo reordenamento, a matriz adquire um formato bloco diagonal, que facilita a decomposição das bases futuras e evita, desta forma, o preenchimento (*fill-in*) da matriz.

Através dos experimentos numéricos descritos na Seção 5, concluímos que a estratégia B_1 apresentou melhores resultados em comparação com as estratégias B_2 e B_3, pois, por exemplo, para 10000 iterações, uma das respostas obtidas foi que o máximo de elementos não nulos na decomposição de B_1 foi 416, enquanto que para B_2 foi 648 e para B_3 foi 491. Como a base possui 2304 elementos, esses valores correspondem, respectivamente, a 82%, 72% e 79% de esparsidade. Podemos observar que a estratégia B_3 assume uma posição intermediária entre as estratégias B_1 e B_2.

Além disso, a utilização de $lu(B_1; 0{,}1)$ geralmente apresenta melhores resultados se comparados com $lu(B_1; 1)$ e com $lu(B_1; 0{,}01)$, obtendo uma diferença maior entre os números de elementos não nulos da decomposição da matriz reordenada B1 e a não reordenada. Desta forma, B_1 contribui para a redução de tempo computacional, podendo resolver problemas de otimização linear de maneira mais eficiente.

Concluímos que o reordenamento estático das colunas da matriz de restrições apresenta várias vantagens computacionais:

- A base inicial, descrita na Seção 4 é triangular para a solução inicial proposta. Desta forma, as decomposições se tornam baratas, em termos computacionais, inclusive a primeira decomposição, que não requer nenhuma operação de ponto flutuante. Assim, a matriz reordenada se torna bloco diagonal, o que evita seu preenchimento durante a decomposição.
- O reordenamento estático leva a decomposições mais esparsas, conforme indicam as tabelas dos experimentos, e a atualização de decomposições extremamente baratas.

- O reordenamento inicial também não tem custo de inicialização e tampouco de atualização, pois como é estático, podemos obter uma decomposição esparsa para qualquer base, sem esforço computacional para ordenar as colunas.
- Por este reordenamento, a decomposição *LU* esparsa pode ser facilmente integrada a outros códigos já existentes, que não estejam explorando a estrutura específica da matriz. O *software* GLPK – *Gnu Linear Programming Kit*, por exemplo, consiste em uma biblioteca de programação linear com código fonte aberto, portanto, se torna mais simples a tarefa de integrá-lo à idéia proposta de reordenamento estático neste trabalho (este *software* pode ser encontrado em www.gnu.org/software/glpk/glpk.html).
- A Tabela 6 apresenta resultados estáveis, onde o erro absoluto entre a matriz obtida desfazendo-se repetidamente toda a decomposição *LU* e a base reordenada resultante das iterações do método simplex é totalmente aceitável, indicando desta forma a robustez do método proposto, o que pode dispensar decomposições periódicas da base como nos métodos tradicionais (Bartels, 1969).
- Esta última conclusão implicaria que por este método de atualização nenhuma decomposição *LU* necessita ser calculada durante toda a execução do método simplex se partimos da base inicial da Seção 4.

Pode ser proposta uma extensão deste problema para um modelo mais completo, como por exemplo, considerar o problema *setup time* (Trigeiro *et al.*, 1989), que consiste em encontrar uma solução factível para problemas de dimensionamento de lotes com capacidade limitada que considere tempo de preparação.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo e pelo CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico.

Referências Bibliográficas

(1) Arenales, M.N.; Morábito, R. & Yanasse, H. (1997). *O problema de Corte e Empacotamento e Aplicações Industriais*. XX Congresso Nacional de Matemática Aplicada e Computacional, Gramado RS, 8 a 12 de setembro.

(2) Bartels R.H. (1969). A Stabilization of the Simplex Method. *Numerical Mathematics*, **16,** 414-434.

(3) Chvátal, V. (1983). *Linear Programming.* San Francisco. W.H. Freeman and Company.

(4) Gilmore, P.C. & Gomory, R.E. (1961). A linear programming approach to the cutting stock problem. *Operations Research*, **9**, 848-859.

(5) Gilmore, P. & Gomory, R. (1965). Multistage cutting stock problems of two and more dimensions. *Operation Research*, **14,** 1045-1074.

(6) Gramani, M.C.N. (2001). Otimização do Processo de Cortagem acoplado ao planejamento da produção. Tese de Doutorado, Densis-Unicamp.

(7) Luenberger, D.G. (1984). *Linear and Nonlinear Programming*. Addison-Wesley.

(8) Marques, F.P. (2000). O Problema da Mochila Compartimentada. Dissertação de Mestrado, ICMC-USP.

(9) Martello, S. & Toth, P. (1989). *Knapsack Problems: Algorithms and Computer Implementations.* Wiley.

(10) Nonas, S.L. & Thorstenson, A. (2000). A combined cutting-stock and lot-sizing problem. *Operation Research*, **120**(2), 327-342.

(11) Trigeiro, W.; Thomas, L.J. & Mcclain, J.O. (1989). Capacited lot sizing with setup times. *Management Science*, **35**(3), 353-366.

(12) Wagner, H.M. & Whitin, T.M. (1958). Dynamic version of the Economic Lot size Model. *Management Science*, **5**(1), 89-96.