Detecção de Linhas Redundantes em Problemas de Programação Linear de Grande Porte¹

D.C. SILVA², A.R.L. OLIVEIRA³, Departamento de Matemática Aplicada, IMECC - Instituto de Matemática, Estatística e Computação Científica, UNICAMP - Universidade Estadual de Campinas, 13083-859 Campinas, SP, Brasil.

Resumo. A presença de linhas redundantes na matriz de restrições não é incomum em problemas reais de grande porte. Se o método de solução adotado for o simplex, existem procedimentos de fácil implementação que contornam este problema. O mesmo se aplica quando métodos de pontos interiores são adotados e os sistemas lineares resultantes são resolvidos por métodos diretos. No entanto, existem problemas de grande porte cuja única forma possível de solução é resolver os sistemas lineares por métodos iterativos. Nesta situação as linhas redundantes geram uma matriz singular e os métodos iterativos não convergem. A única alternativa viável consiste em detectar tais linhas e eliminá-las antes da aplicação do método. Este trabalho tem como objetivo apresentar um procedimento eficiente de detecção de linhas redundantes.

Palavras-chave. Programação linear, linhas redundantes, método de pontos interiores.

1. Introdução

Desde o surgimento dos métodos de pontos interiores para programação linear, códigos computacionais baseados nessas idéias vêm se apresentando como alternativas eficientes para solução de problemas de grande porte [1, 5, 7, 8, 10, 11]. Cada iteração de um método de pontos interiores envolve a solução de um ou mais sistemas lineares [7, 8, 9]. Usualmente métodos diretos são utilizados para resolver estes sistemas. No entanto, por limitações de tempo e memória seu uso torna-se proibitivo em muitos problemas de grande porte, fazendo com que abordagens iterativas sejam mais adequadas.

Para que a opção por métodos iterativos seja bem sucedida é necessário detectar e eliminar todas as linhas redundantes da matriz de restrições, pois de outra forma é obtido um sistema linear singular e os métodos iterativos não convergem. Em [2] é proposta uma técnica sofisticada baseada na decomposição LU de uma base e sua atualização para encontrar todas as linhas redundantes de uma matriz de restrições. Este é o ponto de partida do presente trabalho.

¹Agradecimentos ao CNPq e à FAPESP pelo apoio financeiro.

²silvadc@yahoo.com.br

³aurelio@ime.unicamp.br

A organização deste artigo é a que vem a seguir. Na Seção 2 apresentamos um algoritmo para detecção de linhas redundantes. Na Seção 3 propomos uma implementação eficiente deste algoritmo. Na Seção 4 apresentamos os resultados computacionais. E, finalmente, na Seção 5, nossas conclusões e propostas de trabalhos futuros.

2. O Algoritmo

Considere o seguinte problema de programação linear em sua forma padrão

$$\begin{aligned} & \text{Minimizar} \quad \sum_{j=1}^n c_j x_j \\ \text{s.a} \quad & \sum_{j=1}^n a_{ij} x_j = b_i \quad (i = 1, 2, \dots, m) \\ & x_j \geq 0 \quad (j = 1, 2, \dots, n), \end{aligned} \tag{2.1}$$

onde $A = [a_{ij}] \in \mathbb{R}^{m \times n}$; $x = [x_j]$, $c = [c_j] \in \mathbb{R}^n$ e $b = [b_i] \in \mathbb{R}^m$.

Podemos detectar as linhas redundantes da matriz de restrições A do problema acima (2.1) através da eliminação Gaussiana. Utilizando operações e permutações elementares sobre as linhas de A , a reduzimos para a seguinte forma

$$\begin{bmatrix} x & : & : & : & : & : \\ 0 & x & : & : & : & : \\ 0 & 0 & x & : & : & : \\ 0 & 0 & 0 & x & : & : \\ 0 & 0 & 0 & 0 & x & : \end{bmatrix}. \tag{2.2}$$

Se no decorrer deste processo surgirem linhas nulas na matriz transformada (2.2) estas correspondem as linhas redundantes. Observe que, ao atualizar o vetor b , ou seja, ao aplicar as mesmas operações e permutações realizadas sobre as linhas de A neste vetor, as componentes deste referentes às linhas nulas de (2.2) também devem ser iguais a zero, caso contrário, o problema é infactível.

A desvantagem deste procedimento é que as operações sobre as linhas modifica toda a matriz A , além da grande quantidade de preenchimento que pode ser gerada durante o procedimento, interferindo na esparsidade e estabilidade numérica do método [2].

Alternativamente, podemos utilizar uma matriz base B , a qual é gerada a partir de colunas da matriz de restrições. Para isso, adicionamos variáveis artificiais ao problema (2.1) obtendo o problema equivalente

$$\begin{aligned} & \text{Minimizar} \quad \sum_{j=1}^n c_j x_j \\ \text{s.a} \quad & \sum_{j=1}^n a_{ij} x_j + x_{n+i} = b_i \quad (i = 1, 2, \dots, m) \\ & 0 \leq x_{n+i} \leq 0 \quad (i = 1, 2, \dots, m) \\ & x_j \geq 0 \quad (j = 1, 2, \dots, n), \end{aligned} \tag{2.3}$$

onde $A = [a_{ij}] \in \mathbb{R}^{m \times n}$; $x = [x_j]$, $c = [c_j] \in \mathbb{R}^n$; $b = [b_i] \in \mathbb{R}^m$ e $x_{n+1}, x_{n+2}, \dots, x_{n+m}$ são variáveis artificiais.

Observe que os problemas (2.1) e (2.3) são equivalentes, pois as variáveis artificiais são fixadas em zero.

Seja S o conjunto de índices i tais que x_{n+i} é uma variável básica, (inicialmente todas as variáveis artificiais são básicas, logo a matriz base inicial é a matriz identidade de ordem m). Apresentamos então, o algoritmo para detecção de linhas redundantes.

2.1. Algoritmo para Detecção de Linhas Redundantes

Entradas: A , B e S .

Repita o procedimento abaixo até que todos os índices pertencentes a S tenham sido utilizados.

- [1] Escolha um índice $k \in S$.
- [2] Resolva o sistema $B^t r = e_k$, onde e_k é a k -ésima coluna da matriz identidade.
- [3] Efetue o produto $r^t A$. Se $r^t A$ for um vetor nulo, volte para o passo 1. Do contrário, existe uma variável não básica x_j tal que $r^t a_{.j} \neq 0$, onde $a_{.j}$ é a j -ésima coluna de A . Substitua a k -ésima coluna de B por $a_{.j}$, substitua x_{n+k} por x_j na base e retorne para o passo 1.

Note que, ao fim deste procedimento, algumas variáveis artificiais x_{n+k} podem permanecer na base. Esta permanência indica que a k -ésima linha de A é uma linha redundante e pode ser retirada do problema.

3. Proposta de Implementação

O algoritmo apresentado na seção anterior é bastante conhecido [4]. Porém sua implementação direta pode ter um custo computacional muito elevado. Visando a redução deste custo computacional e a eficácia do algoritmo propomos uma implementação eficiente para este, tomando como base as idéias expostas em [2], as quais serão discutidas a seguir.

Os principais fatores que interferem no custo computacional do algoritmo para detecção de linhas redundantes é a dimensão da base inicial e o número de elementos não nulos contidos nesta. Podemos reduzir estes fatores através de algumas observações.

Primeiramente, observando se há alguma linha nula (linha na qual todos elementos são iguais a zero) na matriz de restrições. Se sim, estas devem ser declaradas redundantes e retiradas do problema.

Posteriormente, verificar se há alguma coluna única (coluna que contém apenas um elemento não nulo) na matriz de restrições. Claramente, a linha na qual esta coluna ocorre é linearmente independente das demais e pode ser retirada temporariamente do problema e não participar do procedimento de verificação de linhas

redundantes. Observe que, quando uma linha é removida, novas colunas únicas podem ser geradas e mais linhas podem ser removidas, o que reduz a dimensão da base inicial.

Outro fator determinante no custo computacional do algoritmo é o número de variáveis artificiais na base inicial, uma vez que, o número de iterações depende desta quantidade. Em [2] é proposta uma heurística cujo objetivo é construir uma base inicial com o maior número de colunas estruturais possível. A seguir, apresentaremos tal heurística. Note que a heurística deve ser aplicada somente após realizados os procedimentos descritos acima.

3.1. Heurística de Andersen

Inicialmente, o número de elementos não nulos em cada linha e coluna da matriz de restrições é calculado. Associe uma variável artificial à linha mais densa. Remova temporariamente esta linha do problema e atualize a contagem de elementos não nulos em cada coluna. Caso ocorra alguma coluna única, atribua a linha na qual ela ocorre à variável básica estrutural correspondente. Repita este procedimento até que cada linha esteja associada a uma variável (artificial ou estrutural). Através desta construção obtemos uma base inicial triangular superior e não singular.

3.2. Forma produto

Em todo o algoritmo de detecção de linhas redundantes, a principal operação de cada iteração consiste na resolução de um sistema linear do tipo $B^t r = e_k$. Portanto, é imprescindível manter uma representação da base (B), capaz de contribuir com a eficiência do algoritmo. Optamos por uma representação inspirada na forma produto da inversa [6].

A forma produto da inversa, foi proposta por Dantzig e Orchard-Hays ao observarem que, em uma troca de base, a inversa da nova base pode ser obtida a partir da inversa da base atual por uma matriz de transformação elementar.

Desta forma, considere E uma matriz de transformação elementar de ordem m , com uma coluna não trivial η na posição k e $B = [b_1, \dots, b_m]$ a matriz base de uma dada iteração. Suponha que a j -ésima variável não básica x_j tenha sido selecionada para substituir a k -ésima variável básica. Denotando $\tilde{B}$ como a nova base, tem-se

$$\tilde{B} = [b_1, \dots, b_{k-1}, a_{.j}, b_{k+1}, \dots, b_m],$$

onde $a_{.j}$ é a coluna da matriz de restrições relacionada a variável x_j .

Sendo $a_{.j}$ um vetor m -dimensional, é possível escrevê-lo como a combinação linear das colunas de B

$$a_{.j} = \sum_{i=1}^m v_i b_i = Bv. \quad (3.1)$$

Se $v_k \neq 0$ então b_k pode ser expresso de acordo com a equação (3.1)

$$b_k = \frac{1}{v_k} a_{.j} - \sum_{i \neq k} \frac{v_i}{v_k} b_i.$$

Definindo o vetor

$$\eta = \left[-\frac{v_1}{v_k}, \dots, -\frac{v_{k-1}}{v_k}, \frac{1}{v_k}, -\frac{v_{k+1}}{v_k}, \dots, -\frac{v_m}{v_k} \right]^t, \quad (3.2)$$

tem-se que $b_k = \tilde{B}\eta$.

Sendo E uma matriz de transformação elementar cuja coluna não trivial é dada por η (3.2), temos

$$B = \tilde{B}E. \quad (3.3)$$

Da equação (3.3) temos

$$\tilde{B} = BE^{-1}.$$

Sendo assim, após p atualizações da matriz base temos

$$B = B_0 E_1^{-1} E_2^{-1} \dots E_p^{-1},$$

onde

B_0 , é a base inicial;

$$E_i = [e_1, \dots, e_{k-1}, \eta_{ki}, e_{k+1}, \dots, e_m]^t;$$

$$\eta_{ki} = \left[-\frac{v_1}{v_k}, \dots, -\frac{v_{k-1}}{v_k}, \frac{1}{v_k}, -\frac{v_{k+1}}{v_k}, \dots, -\frac{v_m}{v_k} \right]^t.$$

Portanto, o sistema $B^t r = e_k$ pode ser resolvido em duas etapas:

$$(E_1^{-1} E_2^{-1} \dots E_p^{-1})^t y = e_k. \quad (3.4)$$

$$B_0^t r = y,$$

onde (3.4) pode ser expresso como $y^t = e_k (E_p \dots E_2 E_1)$.

Em [2], é proposto que o sistema $B^t r = e_k$ seja resolvido através da decomposição LU, a qual é atualizada através da atualização de Bartels-Golub. Porém, optamos pela forma produto, uma vez que a base é atualizada poucas vezes, em no máximo o número de variáveis artificiais presentes na base inicial. Neste caso, a utilização de uma técnica de implementação relativamente simples, como a proposta no presente trabalho, pode tornar o algoritmo mais eficiente pois o esforço e o tempo computacional serão menores.

Outro ponto favorável é a utilização da estrutura triangular da base inicial. Através de algumas modificações propostas em [2], as quais visam a melhora do procedimento de decomposição e atualização LU, a base inicial pode perder sua forma triangular sendo necessária uma nova decomposição LU. Em nossa proposta, a forma triangular da base inicial é preservada e com isto temos $B = LU = IU$, onde I é a matriz identidade e U uma matriz triangular superior conhecida. Assim, resolvemos o sistema $U^t r = e_k$ por meio da resolução de um sistema triangular superior e produto de matrizes elementares por vetores.

4. Resultados Computacionais

Nesta seção apresentaremos os experimentos numéricos realizados e seus respectivos resultados.

Para avaliar o desempenho do algoritmo de detecção de linhas redundantes, o integramos às rotinas de préprocessamento do código PCx modificado [5, 10, 11], o qual utiliza o método de pontos interiores preditor corretor para solucionar os problemas, adotando a tolerância 10^{-8} para seus critérios de convergência (primal, dual e factibilidade).

Primeiramente, são identificadas e retiradas temporariamente do problema as colunas únicas assim como as linhas onde elas ocorrem. A seguir, é aplicada a heurística descrita na Subseção 3.1 e construída a base inicial. A escolha dos índices pertencentes ao conjunto S , passo 1 do algoritmo de detecção de linhas redundantes (ver Seção 2), foi feita em ordem decrescente. E para representação e atualização da base, assim como para a resolução dos sistemas lineares presentes no algoritmo, foi utilizada a forma produto, descrita na Subseção 3.2.

Devido aos erros de arredondamento no passo 3 do algoritmo substituímos a verificação de redundância $r^t a_{.j} \neq 0$ por $|r^t a_{.j}| \leq \varepsilon$. Onde ε é um parâmetro a ser escolhido. Utilizamos $\varepsilon = 10^{-9}$. Caso haja mais de uma coluna tal que $|r^t a_{.j}| \leq 10^{-9}$, optamos pela coluna mais esparsa.

A fim de verificar a interferência da heurística de Andersen (Subseção 3.1) no desempenho do algoritmo, também implementamos o procedimento descrito acima, alterando a heurística da seguinte forma; ao invés de associarmos variáveis artificiais as linhas mais densas, estas foram associadas as linhas mais esparsas. A base inicial gerada permanece triangular superior e não singular.

Por fim, como parâmetro de comparação integramos o procedimento descrito em [10, 11] ao préprocessamento do código PCx modificado. Este é um procedimento pouco sofisticado que detecta as linhas redundantes por meio de uma decomposição LU retangular da matriz de restrições, após eliminar recursivamente as linhas de elementos de colunas únicas.

As implementações foram feitas em linguagem C e testadas em uma plataforma Intel(R) Pentium(R) D 3.40GHz e com 2Gb de memória RAM, em Linux, utilizando compiladores gcc e gfortran.

Denominamos as rotinas de detecção de linhas redundantes implementadas neste trabalho como, LU retangular (procedimento descrito em [10, 11]), Redundante e Redundante Modificado (procedimentos descritos no presente trabalho). A rotina Redundante Modificado é o procedimento no qual a heurística de Andersen é modificada.

Os problemas usados como testes são apresentados na Tabela 1. Todos os problemas são de domínio públicos e podem ser encontrados em NETLIB⁴, MISC⁵ e QAP [3]. Optamos por alguns dos problemas de maior porte destas coleções, uma vez que, o foco deste trabalho são problemas de grande porte.

Na Tabela 2 são apresentados dados sobre a dimensão destes problemas após

⁴Netlib collection LP test sets, Netlib lp repository, Online at <http://www.netlib.org/lp/data>.

⁵Miscellaneous LP models, Hungarian Academy of Sciences OR Lab, Online at <http://www.sztaki.hu/meszaros/public-ftp/lptestset/mist>.

Tabela 1: Problemas testes

Problema	Linhas	Colunas	Nnulos	Coleção
PDS-30	49941	158489	340635	MISC
PDS-40	66844	217531	466800	MISC
PDS-50	83060	275814	590833	MISC
PDS-60	99431	336421	719557	MISC
PDS-70	114944	390005	833465	MISC
PDS-80	129181	434580	927826	MISC
PDS-90	142823	475448	1014136	MISC
DFL001	6701	12230	35632	NETLIB
PDS-20	33874	108175	232647	NETLIB
KRA30A	18059	85725	337920	QAP
NUG06-3rd	3972	4686	22032	QAP
NUG07-3rd	9422	12691	61544	QAP
NUG12	3192	8856	38304	QAP
NUG15	6330	22275	94950	QAP
NUG30	21899	141405	568320	QAP
STE36A	27683	131076	512640	QAP

o préprocessamento e a detecção de linhas redundantes. Nesta tabela, as colunas PLinhas e PColunas representam, respectivamente, o número de linhas e colunas após a aplicação das rotinas de préprocessamento do código PCx juntamente com os procedimentos de detecção de linhas redundantes. Já as colunas PRNnulos1 e PRNnulos2, representam, respectivamente, o percentual de redução de elementos não nulos em relação à formulação original dos problemas quando são utilizadas somente as rotinas de préprocessamento do código PCx, e quando juntamente a estas rotinas é utilizado algum dos procedimentos de detecção de linhas redundantes descritos neste artigo. Por fim, a coluna LR representa o número de linhas redundantes identificadas.

Através desta tabela, podemos perceber a importância da detecção e remoção das linhas redundantes em problemas de programação linear de grande porte. A maioria dos problemas teste apresentaram linhas redundantes e em alguns casos (NUG06-3rd, NUG07-3rd, NUG12 e NUG15) foi identificado um número elevado de linhas redundantes, as quais geram uma série de complicações no processo de solução. Enfatizando que as técnicas tradicionais de préprocessamento não são suficientes em todos os contextos. Além disso, com a detecção de linhas redundantes reduzimos em média 3,5% o número de elementos não nulos. Redução esta, igual ou superior a 10% para os problemas NUG06-3rd, NUG12 e NUG15. Enquanto que aplicando somente as rotinas de préprocessamento do código PCx que não detecta todas as linhas redundantes, esta redução é em média de 0,75%.

A Tabela 3 apresenta a dimensão da matriz base (B) e o número de iterações do algoritmo de detecção de linhas redundantes quando este é implementado direta-

Tabela 2: Dados dos Problemas Testes após o Preprocessamento

Problema	PLinhas	PColunas	PRNnulos1	PRNnulos2	LR
PDS-30	47957	156042	2%	2%	11
PDS-40	64265	214385	2%	2%	11
PDS-50	80328	272513	1%	2%	11
PDS-60	96503	332862	1%	1%	11
PDS-70	111885	386238	1%	1%	11
PDS-80	126109	430800	1%	1%	11
PDS-90	139741	471538	1%	1%	11
DFL001	5971	12143	1%	1%	13
PDS-20	32276	106180	2%	3%	11
KRA30A	18059	85725	0%	0%	0
NUG06-3rd	3540	4686	0%	11%	432
NUG07-3rd	8594	12691	0%	9%	828
NUG12	2794	8856	0%	12%	398
NUG15	5698	22275	0%	10%	632
NUG30	21899	141405	0%	0%	0
STE36A	27683	131076	0%	0%	0
Média percentual de redução:			0,75 %	3,50 %	

mente e quando a heurística de Andersen (com e sem alterações) é utilizada. Nesta tabela, a coluna B1 indica a dimensão da base caso o algoritmo fosse implementado diretamente. B2 representa a dimensão da base quando a heurística (com e sem alterações) é utilizada no processo de construção da base. Já a coluna PR representa o percentual de redução de B2 em relação a B1. Por fim, as colunas VA1 e VA2 mostram, respectivamente, o número de colunas artificiais presentes na base inicial quando a heurística é aplicada diretamente e com modificações. Observe que, o número de colunas artificiais presentes na base inicial corresponde ao número de iterações do algoritmo.

Podemos observar que houve uma redução significativa nos fatores dimensão da base e números de colunas artificiais presentes na base inicial, o que mostra que a heurística proposta (mesmo com as alterações realizadas neste trabalho) de fato contribui para a eficácia do algoritmo.

Por fim, as Tabelas 4 e 5 apresentam os resultados relevantes de tempo.

Na Tabela 4 são apresentados dados sobre o tempo de preprocessamento. A coluna TP representa o tempo (em segundos) utilizado pelas rotinas de preprocessamento do código PCx. TLR1, TLR2 e TLR3 representam, respectivamente, o tempo (em segundos) utilizado pelas rotinas LU Retangular, Redundante e Redundante Modificado para identificar as linhas redundantes. Já as colunas TTP1, TTP2 e TTP3 representam, respectivamente, a soma dos tempos de preprocessamento e detecção de linhas redundantes quando são utilizadas as rotinas LU Retangular, Redundante e Redundante Modificado.

Tabela 3: Comparação entre dimensões e número de iterações

Problema	B1	B2	PR	VA1	VA2
PDS-30	47968	44795	6,61 %	13	44
PDS-40	64276	60030	6,60 %	13	44
PDS-50	80339	75070	7,06 %	13	44
PDS-60	96514	90226	6,51 %	13	44
PDS-70	111896	104724	6,41 %	13	44
PDS-80	126120	118342	6,17 %	13	44
PDS-90	139752	131499	5,90 %	13	44
DFL001	5984	4275	28,56 %	15	19
PDS-20	32287	30088	6,81 %	13	44
KRA30A	18059	5009	72,26 %	165	167
NUG06-3rd	3972	2172	45,31 %	949	1043
NUG07-3rd	9422	5012	46,80 %	2142	2284
NUG12	3192	3192	0,00 %	1664	1664
NUG15	6330	6330	0,00 %	3272	3272
NUG30	21899	8849	59,60 %	293	323
STE36A	27683	6226	77,51 %	172	173
Média percentual de redução:			18,84 %		

A Tabela 5 apresenta dados sobre o tempo de solução do problema de programação linear. Nesta tabela, as colunas TT1, TT2 e TT3 representam, respectivamente, o tempo total (em segundos) utilizado para solucionar os problemas quando identificamos as linhas redundantes através das rotinas LU Retangular, Redundante e Redundante Modificado. IT1, IT2 e IT3 representam, respectivamente, o número de iterações do método de pontos interiores (disponível no código PCx modificado) necessário para solucionar os problemas quando são utilizadas as rotinas LU Retangular, Redundante e Redundante Modificado.

Através destas tabelas, podemos verificar que a rotina tradicional de detecção de linhas redundantes (LU Retangular) mostrou-se pouco eficiente. Apresenta elevados tempos de preprocessamento (Tabela 4) e para a maioria dos problemas, o tempo utilizado para solucionar os problemas é maior quando este procedimento é utilizado (Tabela 5), comprometendo a eficiência do método de solução.

Tabela 4: Resultados de tempo de processamento para os problemas teste

Problema	TP	TLR1	TLR2	TLR3	TTP1	TTP2	TTP3
PDS-30	0,28	42,71	19,19	19,45	42,99	19,47	19,73
PDS-40	0,29	78,99	35,33	35,62	79,18	35,62	35,91
PDS-50	0,37	134,58	55,92	56,08	134,95	56,29	56,45
PDS-60	1,16	213,72	82,56	83,12	214,88	83,72	84,28
PDS-70	0,64	308,85	111,42	111,35	309,49	112,06	111,99
PDS-80	0,24	405,13	141,11	142,27	405,37	141,35	142,51
PDS-90	0,27	531,87	172,08	170,56	532,14	172,35	170,83
DFL001	0,05	0,25	0,18	0,18	0,30	0,23	0,23
PDS-20	0,10	18,38	8,82	9,04	18,48	8,92	9,14
KRA30A	0,14	0,16	1,76	1,75	0,30	1,90	1,89
NUG06-3rd	0,00	1,54	0,38	0,37	1,54	0,38	0,37
NUG07-3rd	0,00	17,34	2,56	2,46	17,34	2,56	2,46
NUG12	0,01	5,17	1,31	1,23	5,18	1,32	1,24
NUG15	0,00	46,45	6,37	6,20	46,45	6,37	6,20
NUG30	0,25	0,48	5,20	5,39	0,73	5,45	5,64
STE36A	0,15	0,26	3,16	3,11	0,41	3,31	3,26
Média	0,25	112,83	35,86	33,01	56,08	40,71	40,76

Em contrapartida, nossas propostas (Redundante e Redundante Modificado) apresentaram um desempenho bastante satisfatório. Em comparação com a rotina LU Retangular, mostraram-se em média três vezes mais rápidas, chegando a ser sete vezes mais rápidas no caso do problema NUG07-3rd. Entretanto, houve uma pequena queda de desempenho quando o problema não apresenta linhas redundantes (KRA30A, NUG30 e STE36A). Isto é devido ao fato de que a ausência de linhas redundantes é facilmente identificada pela LU-Retangular ao contrário de nossos procedimentos onde é necessário realizar todas as iterações para identificar tal ausência. Entretanto, esta é uma perda pequena comparada com os ganhos nos casos em que estas linhas estão presentes. Em relação ao tempo de solução, o método de pontos interiores necessita de um tempo menor para solucionar a maioria dos problemas teste quando nossas rotinas de detecção de linhas redundantes são utilizadas, reduzindo o tempo total de solução conforme apresentado na Tabela 5. Quanto ao número de iterações realizadas para solucionar os problemas teste não houve grandes variações independente do procedimento de detecção de linhas redundantes adotado. Exceto para o problema NUG15. Neste caso, quando nossas rotinas são utilizadas, o método de pontos interiores necessita de um número de iterações bem maior em relação ao número de iterações realizadas quando a LU Retangular é aplicada. Este fato explica o desempenho superior da LU retangular em relação às rotinas Redundante e Redundante Modificado neste problema.

Por fim, ao compararmos os procedimentos descritos neste trabalho, podemos perceber que a rotina Redundante Modificado apresenta a maioria dos melhores

Tabela 5: Resultados de tempo de solução para os problemas teste

Problema	TT1	TT2	TT3	IT1	IT2	IT3
PDS-30	252,93	226,33	221,94	37	36	39
PDS-40	427,79	400,54	377,56	33	35	39
PDS-50	721,67	628,83	571,98	36	36	36
PDS-60	1069,95	900,53	846,49	35	37	35
PDS-70	1404,97	1241,13	1147,75	34	34	37
PDS-80	1916,72	1665,95	1671,08	37	37	37
PDS-90	2363,54	2065,21	2119,66	37	37	37
DFL001	25,94	25,89	26,64	16	16	16
PDS-20	200,50	191,66	144,90	42	42	42
KRA30A	1207,37	1208,97	1208,96	41	41	41
NUG06-3rd	294,33	194,85	321,87	8	7	8
NUG07-3rd	13888,83	6483,93	15397,45	10	8	8
NUG12	148,13	156,35	158,56	13	13	13
NUG15	2217,22	11279,65	11092,39	13	41	41
NUG30	3164,81	3169,53	3169,72	41	41	41
STE36A	787,82	790,72	790,67	40	40	40
Média	1880,78	1914,38	1218	29,56	31,31	31,88

tempos totais de solução (Tabela 5).

5. Conclusão

Nossos procedimentos mostram-se eficientes em termos de tempo computacional, fator de extrema importância quando estamos tratando de pré-processamento em programação linear, ao serem comparados com um procedimento de detecção de linhas redundantes baseado na eliminação Gaussiana. A diferença de desempenho entre os procedimentos aumentou de acordo com a dimensão do problema, enfatizando que as propostas deste trabalho podem ser alternativas eficazes para a resolução do problema em questão.

Além disso, nossos procedimentos interferiram de forma positiva no processo de solução dos problemas. Ao aplicá-los para a detecção de linhas redundantes, reduzimos o tempo utilizado pelo método de pontos interiores para solucionar os problemas, na maioria dos casos, conduzindo a uma redução do tempo total de solução.

Como trabalho futuro, propomos a implementação do procedimento de detecção de linhas redundantes exatamente como descrito no artigo de Andersen [2] para uma possível comparação entre este e os procedimentos descritos neste trabalho.

Abstract. The presence of dependent rows in the constraint matrix is frequent in real large-scale problems. If the solution method adopted is the simplex method, there are efficient procedures easy to implement that circumvent this problem. The same applies when interior point methods are adopted and the resulting linear systems are solved for directed methods. However, there are large-scale problems whose only possible approach consists in solving the linear systems by iterative methods. In this situation, the dependent rows create a singular matrix and the iterative method does not converge. The only viable alternative is to find and remove these rows before applying the method. This paper proposes an efficient implementation of a procedure for dependent rows detection.

Referências

- [1] I. Adler, M.G.C. Resende, G. Veiga, N. Karmakar, An implementation of Karmarkar's algorithm for linear programming, *Mathematical Programming*, **44** (1989), 297–335.
- [2] E.D. Andersen, Finding all linearly dependent rows in large-scale linear programming, *Optimization Methods and Software*, **6** (1995), 219–227.
- [3] R.S. Burkard, S. Karisch, F. Rendl, Qaplib - a quadratic assignment problem library, *European Journal of Operations Research*, **55** (1991), 115–119.
- [4] V. Chvátal, "Linear Programming", W.H. Freeman and Company, New York, 1983.
- [5] J. Czyzyk, S. Mehrotra, M. Wagner, S.J. Wright, PCx an interior point code for linear programming, *Optimization Methods and Software*, **11**, No. 2 (1999), 397–430.
- [6] G.B. Dantzig, W. Orchard-Hays, The product form for the inverse in the simplex method, *Mathematical Tables and Others Aids to Computation*, **8**, No. 46 (1954), 64–67.
- [7] J. Gondzio, Multiple centrality corrections in a primal-dual methods for linear programming, *Computation Optimization and Applications*, **6** (1996), 137–156.
- [8] I.J. Lustig, R.E. Marsten, D.F. Shanno, On implementing Mehrotra's predictor-corrector interior point method for linear programming, *SIAM Journal on Optimization*, **2** (1992), 435–449.
- [9] S. Mehrotra, On implementation of a primal-dual point method, *SIAM Journal on Optimization*, **2** (1992), 575–601.
- [10] A.R.L. Oliveira, D.C. Sorensen, A new class of preconditioners for large-scale systems from interior point methods for linear programming, *Linear Algebra Its Applications*, **394** (2005), 1–24.
- [11] M.I. Velazco, A.R.L. Oliveira, F.F. Campos, A note on hybrid preconditioners for large scale normal equations arising from interior-point methods, *Optimization Methods and Software*, **25**, No. 2 (2010), 321–332.