# DETECÇÃO DE LINHAS REDUNDANTES EM PROBLEMAS DE PROGRAMAÇÃO LINEAR DE GRANDE PORTE

**Daniele Costa Silva**
Depto de Matemática Aplicada, IMECC, UNICAMP
13083-859, Campinas, SP
silvadc@yahoo.com.br

**Aurelio R. L. de Oliveira**
Depto de Matemática Aplicada, IMECC, UNICAMP
13083-859, Campinas, SP
aurelio@ime.unicamp.br

**Carla T. L. da Silva Ghidini**
Depto de Matemática Aplicada, IMECC, UNICAMP
13083-859, Campinas, SP
carla@ime.unicamp.br

## RESUMO

A presença de linhas redundantes na matriz de restrições não é incomum em problemas reais de grande porte. A existência de tais linhas deve ser levada em consideração na solução destes problemas. 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 oriundos dos métodos de pontos interiores por métodos iterativos. Nesta situação as linhas redundantes representam uma dificuldade considerável, pois 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 Matemática. Linhas Redundantes. Método de Pontos Interiores.

## ABSTRACT

The presence of dependents rows in the constraint matrix is very common in real large-scale problems. The existence these rows should be considered in the solution these problems. If we use the simplex method, there are procedures of easy implementation, that circumvent this problem. The same occurs when use interior-point methods and the linear systems arising are solved by direct methods. However, there are large-savel problems whose only possible solution is solve the linear systems from the interior-point methods for iteratives methods. In this situation the dependent rows represent a considerable difficulty, because create a singular matrix and the iteratives methods do not converge. The only alternative is detect this rows and eliminates before the application of the method. This work aims to present en efficient method for detection of dependents rows.
**KEYWORDS.** Mathematical Programming. Dependents Rows. Interior-Point Methods.

# 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 (Adler et al, 1989, Bocanegra et al 2007, Czyzyk et al, 1999, Gondzio, 1996, Lusting et al, 1992, Oliveira e Sorensen, 2005). Cada iteração de um método de pontos interiores envolve a solução de um ou mais sistemas lineares (Gondzio, 1996, Lusting et al, 1992, Mehrotra, 1992). Em aplicações reais estes sistemas quase sempre possuem dimensões elevadas e alto grau de esparsidade.

Usualmente métodos diretos são utilizados para resolver estes sistemas. A abordagem mais utilizada nas implementações existentes é a fatoração de Cholesky no sistema de equações normais (Adler et al, 1989, Czyzyk et al, 1999, Gondzio, 1996, Lusting et al, 1992). 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. Antes da aplicação dos métodos iterativos a um problema de programação linear, este é preprocessado para reduzir sua dimensão, detectar possíveis infactibilidade e aprimorar a estabilidade numérica (Andersen e Andersen 1995, Suhl, 1994, Tomlin e Welch, 1986). No preprocessamento são aplicadas regras, quase sempre simples, que permitem encontrar variáveis com valor fixo, linhas duplicadas entre diversas outras situações.

Embora as técnicas utilizadas nestes trabalhos encontrem algumas linhas redundantes nem todas são necessariamente detectadas por ser este considerado um processo computacionalmente caro. Em (Andersen, 1995) 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 foi o ponto de partida do presente trabalho.

O texto está organizado da seguinte forma: na Seção 2, apresentamos um conhecido algoritmo para detecção de linhas redundantes. Na seção 3, propomos uma implementação deste algoritmo a qual será comparada com a proposta em (Andersen, 1995). Na seção 4, apresentamos os experimentos e resultados computacionais. Finalmente, na Seção 5, estão as conclusões.

# 2 O Algoritmo

Considere o seguinte problema de programação linear em sua forma padrão:

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

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

Podemos detectar as linhas redundantes da matriz de restrições $A$ do problema (1) através da eliminação Gaussiana. Utilizando operações elementares sobre as linhas de $A$, a reduzimos para a seguinte forma:

$$\begin{bmatrix} x & : & : & : & : & : \\ 0 & x & : & : & : & : \\ 0 & 0 & x & : & : & : \\ 0 & 0 & 0 & x & : & : \\ 0 & 0 & 0 & 0 & x & : \end{bmatrix}. \tag{2}$$

Se no decorrer deste processo surgirem linhas nulas na matriz transformada (2) estas correspondem as linhas redundantes. Observe que, ao atualizar o vetor $b$, ou seja, ao aplicar as mesmas operações realizadas sobre as linhas de $A$ neste vetor, as componentes deste referentes às linhas nulas de (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 (Andersen, 1995) .

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 (1) obtendo o problema equivalente:

$$\begin{aligned} & \text{Minimizar} \quad \sum_{j=1}^{n} c_j x_j \\ \text{s.a} \quad & \sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i \quad (i = 1, 2, ..., m) \\ & 0 \leq x_{n+i} \leq 0 \quad (i = 1, 2, .., m) \\ & x_j \geq 0 \quad (j = 1, 2, ..., n) \end{aligned} \tag{3}$$

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

Observe que os problemas (1) e (3) são equivalentes, pois as variáveis artificiais são fixadas em zero.

### 2.1 Algoritmo para Detecção de Linhas Redundantes

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 $m$).

**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. Se não, 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 Subseção 2.1 é bastante conhecido (Chvàtal, 1983). 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 (Andersen, 1995) 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, verificar se há alguma linha nula 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 (Andersen, 1995) é proposta uma heurística cujo objetivo é construir uma base inicial com o maior número de colunas estruturais (colunas da matriz $A$) 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 (Dantzig e Orchard-Hays, 1954) .

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 $\eta$ na posição $k$ e $B = [b_1, ..., b_m]$ 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:

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

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 $B$:

$$a_{.j} = \sum_{i=1}^{m} v_i b_i = Bv. \tag{4}$$

Se $v_k \neq 0$ então $b_k$ pode ser expresso de acordo com a equação (4):

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

Definindo o vetor

$$\eta = [-\frac{v_1}{v_k}, ..., -\frac{v_{k-1}}{v_k}, \frac{1}{v_k}, -\frac{v_{k+1}}{v_k}, ..., -\frac{v_m}{v_k}]^t \tag{5}$$

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

Sendo $E$ uma matriz de transformação elementar cuja coluna não trivial é dada por $\eta$ (5), temos:

$$B = \tilde{B}E. \tag{6}$$

Da equação (6) temos que:

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

Sendo assim, após $p$ atualizações da matriz base temos:

$$B = B_0 E_1^{-1} E_2^{-1} ... E_p^{-1}$$

onde:
$B_0$ é a base inicial;

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

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

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

$$(E_1^{-1} E_2^{-1} ... E_p^{-1})^t y = e_k red. \tag{7}$$
$$B_0^t r = y red.$$

A equação (7) é equivalente a $y^t = e_k(E_p ... E_2 E_1)$.

Em (Andersen, 1995), é proposto que o sistema $B^t r = e_k$ seja resolvido utilizando a decomposição LU, a qual é atualizada através da atualização de Bartels-Golub. Neste trabalho iremos comparar ambos os métodos de solução a fim de verificar qual procedimento é mais eficiente.

## 4 Experimentos 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 preprocessamento do código PCx modificado (Czyzyk et al, 1999, Oliveira e Sorensen, 2005, Velazco eta al, 2010) 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) e o método dos gradientes conjugados precondicionado para resolução dos sistemas lineares.

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. 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 $\varepsilon$ 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.

Como parâmetro de comparação integramos o procedimento descrito em (Andersen, 1995) ao preprocessamento do código PCx modificado.

Por fim, em (Andersen, 1995) após a construção da base inicial é proposto que a base seja permutada de forma que as últimas *va* colunas, onde *va* é o número de variáveis artificiais presentes na base inicial, sejam iguais as *va* últimas colunas da matriz identidade de mesma ordem. Segundo o autor, esta permutação é realizada para tornar a resolução dos sistemas lineares do algoritmo mais eficaz. A fim de verificar a influência dessa permutação na eficiência da detecção de linhas redundantes, implementamos ambos os procedimentos com e sem essa permutação.

As implementações foram feitas em linguagem C e testadas em uma plataforma Intel(R) Core(TM) D 2.93GHz e com 16Gb de memória RAM, em Linux 64Bits, utilizando os compiladores gcc e gfortran.

Denominamos as rotinas de detecção de linhas redundantes implementadas neste trabalho como: LU (procedimento descrito em (Andersen, 1995), LU Modificado (procedimento descrito em (Andersen, 1995) sem a permutação da base inicial), Forma Produto e Forma Produto Modificado (procedimentos descritos neste trabalho com e sem a permutação da base inicial, respectivamente).

Neste trabalho utilizamos 28 problemas testes os quais são apresentados nas Tabelas 1 e 2. Nestas tabelas, as colunas Linhas, Colunas e Nnulos trazem informações sobre a matriz de restrições de cada problema. A coluna Linhas contém o número de linhas dessa matriz, a coluna Colunas o número de colunas e a coluna Nnulos o número de elementos não nulos. Já a coluna Coleção informa a qual coleção o problema pertence. Todos os problemas são de domínio público e podem ser encontrados em NETLIB[1], MISC[2] e QAP (Burkard et al, 1991). Optamos por alguns dos problemas de maior porte destas coleções, uma vez que, o foco deste trabalho são problemas de grande porte.

Tabela 1: Problemas testes

| Problema | Linhas | Colunas | Nnulos | Coleção |
|---|---|---|---|---|
| ELS19 | 4350 | 13186 | 50882 | QAP |
| CHR25a | 8149 | 15325 | 53725 | QAP |
| CHR22b | 5587 | 10417 | 36520 | QAP |
| SCR20 | 5079 | 15980 | 61780 | QAP |
| CRE-b | 9648 | 77137 | 260785 | NETLIB |
| CRE-d | 8926 | 73948 | 246614 | NETLIB |
| KEN-11 | 14694 | 21349 | 49058 | NETLIB |
| KEN-13 | 28632 | 42659 | 97246 | NETLIB |
| KEN-18 | 105127 | 154699 | 358171 | NETLIB |
| PDS-06 | 9881 | 29351 | 63220 | QAP |
| ROU20 | 7359 | 37640 | 152980 | QAP |
| PDS-10 | 16558 | 49932 | 107605 | NETLIB |
| PDS-20 | 33874 | 108175 | 232647 | NETLIB |
| 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 |

[1]Netlib collection LP test sets, Netlib lp repository, Online at http://www.netlib.org/lp/data.

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

Tabela 2: Problemas testes (continuação)

| Problema | Linhas | Colunas | Nnulos | Coleção |
|---|---|---|---|---|
| PDS-100 | 156243 | 514577 | 1096002 | QAP |
| QAP12 | 3192 | 8856 | 38304 | QAP |
| NUG07-3rd | 9422 | 12691 | 61544 | QAP |
| NUG08-3rd | 19728 | 29856 | 148416 | QAP |
| NUG12 | 3192 | 8856 | 38304 | QAP |
| NUG15 | 6330 | 22275 | 94959 | QAP |
| QAP15 | 6330 | 22275 | 94959 | NETLIB |
| STE36a | 27683 | 13076 | 512640 | QAP |

Nas Tabelas 3 e 4, são apresentados dados sobre a dimensão destes problemas após o preprocessamento e a detecção de linhas redundantes. Nestas tabelas , as colunas PLinhas e PColunas contêm, respectivamente, o número de linhas e colunas após a aplicação das rotinas de preprocessamento do código PCx juntamente com os procedimentos de detecção de linhas redundantes. Já na coluna PRNnulos, está o percentual de redução de elementos não nulos em relação à formulação original dos problemas e quando é utilizado algum dos procedimentos de detecção de linhas redundantes. Por fim, a coluna LR contém o número de linhas redundantes identificadas.

Tabela 3: Dados dos Problemas Testes após o Preprocessamento

| Problema | PLinhas | PColunas | PRNnulos | LR |
|---|---|---|---|---|
| ELS19 | 4350 | 13186 | 0% | 0 |
| CHR25a | 8149 | 15325 | 0% | 0 |
| CHR22b | 5587 | 10417 | 0% | 0 |
| SCR20 | 5079 | 15980 | 0% | 0 |
| CRE-b | 5328 | 36382 | 57% | 8 |
| CRE-d | 4094 | 28601 | 65% | 8 |
| KEN-11 | 9964 | 16740 | 25% | 121 |
| KEN-13 | 22365 | 36561 | 17% | 169 |
| KEN-18 | 78538 | 128434 | 18% | 324 |
| PDS-06 | 9145 | 28472 | 5% | 11 |
| ROU20 | 7359 | 37640 | 0% | 0 |
| PDS-10 | 15637 | 48780 | 3% | 11 |
| PDS-20 | 32276 | 106180 | 3% | 11 |
| PDS-30 | 47957 | 156042 | 2% | 11 |
| PDS-40 | 64265 | 214385 | 2% | 11 |
| PDS-50 | 80328 | 272513 | 2% | 11 |
| PDS-60 | 96503 | 332862 | 1% | 11 |
| PDS-70 | 111885 | 386238 | 1% | 11 |
| PDS-80 | 126109 | 430800 | 1% | 11 |
| PDS-90 | 139741 | 471538 | 1% | 11 |
| PDS-100 | 152289 | 498530 | 3% | 11 |
| QAP12 | 2794 | 8856 | 12% | 398 |
| NUG07-3rd | 8594 | 12691 | 9% | 828 |
| NUG08-3rd | 18270 | 29856 | 7% | 1458 |

Tabela 4: Dados dos Problemas Testes após o Preprocessamento (continuação)

| Problema | PLinhas | PColunas | PRNnulos | LR |
|---|---|---|---|---|
| NUG12 | 2794 | 8856 | 12% | 398 |
| NUG15 | 5698 | 22275 | 10% | 632 |
| QAP15 | 5698 | 22275 | 10% | 632 |
| STE36a | 27683 | 13076 | 0% | 0 |
| **Média percentual de redução:** | | | | **10 %** |

Através destas tabelas, podemos perceber a importância do preprocessamento na resolução de problemas de programação linear de grande porte. O número de elementos não nulos foi reduzido em média 10%, redução esta, superior a 50% para os problemas CRE-b e CRE-d. No entanto, as técnicas tradicionais de preprocessamento não são suficientes em todos os contextos. A maioria dos problemas teste apresentaram linhas redundantes e em alguns casos (KEN11, KEN13, KEN18, QAP12, NUG07-3rd, NUG08-3rd, NUG 12, NUG 15 e QAP15) foi identificado um número elevado de linhas redundantes, as quais geram uma série de complicações no processo de solução. Enfatizando a importância e a necessidade da detecção e remoção de linhas redundantes em problemas de programação linear de grande porte.

Nas Tabelas 5 e 6, são apresentados dados sobre o tempo de preprocessamento. Na coluna TP está o tempo (em segundos) utilizado pelas rotinas de preprocessamento do código PCx. As colunas TLRLU, TLRLUM, TLRFP e TLRFPM trazem, respectivamente, o tempo (em segundos) utilizado pelas rotinas LU, LU Modificado, Forma Produto e Forma Produto Modificado para identificar as linhas redundantes.

Tabela 5: Tempo de preprocessamento

| Problema | TP | TLRLU | TLRLUM | TLRFP | TLRFPM |
|---|---|---|---|---|---|
| ELS19 | 0,00 | 0,06 | 0,07 | **0,03** | **0,03** |
| CHR25a | 0,00 | 0,05 | 0,05 | **0,01** | **0,01** |
| CHR22b | 0,00 | 0,04 | 0,05 | **0,01** | **0,01** |
| SCR20 | 0,00 | 0,07 | 0,08 | 0,04 | **0,02** |
| CRE-b | 0,07 | 0,06 | 0,07 | **0,03** | **0,03** |
| CRE-d | 0,06 | 0,06 | 0,07 | **0,03** | **0,03** |
| KEN-11 | 0,00 | **0,21** | 0,24 | **0,21** | 0,22 |
| KEN-13 | 0,02 | **0,72** | 0,85 | 0,84 | 0,94 |
| KEN-18 | 0,10 | **8,26** | 9,98 | 10,45 | 11,22 |
| PDS-06 | 0,01 | 0,29 | 0,23 | **0,19** | 0,21 |
| ROU20 | 0,01 | 0,24 | 0,24 | **0,22** | **0,22** |
| PDS-10 | 0,02 | **0,54** | 0,58 | 0,55 | 0,61 |
| PDS-20 | 0,03 | **2,28** | 2,38 | 2,44 | 2,76 |
| PDS-30 | 0,06 | **4,90** | 5,14 | 5,26 | 6,22 |
| PDS-40 | 0,09 | **8,96** | 9,45 | 9,75 | 11,47 |
| PDS-50 | 0,11 | **13,98** | 14,80 | 15,46 | 17,23 |
| PDS-60 | 0,14 | **20,46** | 21,78 | 22,67 | 25,26 |
| PDS-70 | 0,16 | **27,28** | 29,13 | 30,56 | 34,00 |
| PDS-80 | 0,17 | **33,76** | 37,11 | 38,61 | 42,84 |
| PDS-90 | 0,20 | **40,57** | 45,04 | 47,30 | 51,86 |
| PDS-100 | 0,26 | **46,74** | 53,44 | 56,86 | 60,42 |
| QAP12 | 0,01 | 1,03 | 1,03 | 0,34 | **0,32** |

Tabela 6: Tempo de preprocessamento (continuação)

| Problema | TP | TLRLU | TLRLUM | TLRFP | TLRFPM |
|---|---|---|---|---|---|
| NUG07-3rd | 0,48 | 4,89 | 4,86 | 0,81 | **0,75** |
| NUG08-3rd | 0,00 | 42,55 | 43,04 | 3,74 | **3,48** |
| NUG12 | 0,00 | 3,37 | 3,45 | 0,48 | **0,44** |
| NUG15 | 0,00 | 25,92 | 26,80 | 2,21 | **2,07** |
| QAP15 | 0,05 | 6,96 | 6,75 | **1,51** | 1,55 |
| STE36a | 0,05 | **0,90** | 0,96 | 0,97 | 0,95 |
| **Média:** | 0,07 | 10,54 | 11,34 | **8,98** | 9,83 |

De acordo com as Tabelas 5 e 6 podemos perceber que a permutação proposta em (Andersen, 1995) torna a detecção de linhas redundantes um pouco mais eficiente. As rotinas cuja permutação foi implementada (LU e Forma Produto), apresentam melhores tempos de processamento, mas essa melhora é pouco significativa. No geral a variação entre os tempos de processamento é pequena. A rotina LU é a que detem a maioria dos melhores tempos. No entanto, a rotina Forma Produto apresenta uma melhor média de tempo utilizado para detecção de linhas redundantes, chegando a ser cerca de onze vezes mais rápida que a rotina LU para os problemas NUG08-3rd e NUG15.

Nas Tabelas 7 e 8 são apresentados dados sobre o tempo de solução dos problemas testes. Nestas tabelas, as colunas TPCLU, TPCLUM, TPCFP e TPCFPM representam, respectivamente, o tempo (em segundos) utilizado pelo método de pontos interiores preditor corretor (disponível no código PCx modificado) para solucionar os problemas quando identificamos as linhas redundantes através das rotinas LU, LU Modificado, Forma Produto e Forma Produto Modificado. A coluna IT representa o número de iterações do método de pontos interiores necessário para solucionar os problemas, número este que não se altera com o método de detecção de linhas redundantes.

Tabela 7: Tempo de solução

| Problema | TPCLU | TPCLUM | TPCFP | TPCFPM | IT |
|---|---|---|---|---|---|
| ELS19 | **69,94** | **69,94** | **69,94** | **69,94** | 31 |
| CHR25a | **57,52** | **57,52** | **57,52** | **57,52** | 29 |
| CHR22b | **27,44** | **27,44** | **27,44** | **27,44** | 29 |
| SCR20 | **92,41** | **92,41** | **92,41** | **92,41** | 21 |
| CRE-b | **68,91** | 69,48 | 70,25 | 76,04 | 44 |
| CRE-d | **36,70** | 36,95 | 37,36 | 37,54 | 44 |
| KEN-11 | **11,22** | 11,24 | 11,65 | 12,55 | 23 |
| KEN-13 | **112,20** | 113,13 | 116,73 | 117,93 | 30 |
| KEN-18 | **1522,62** | 1530,22 | 1579,18 | 1589,43 | 41 |
| PDS-06 | 14,14 | **14,10** | 14,43 | 15,12 | 38 |
| ROU20 | **1267,53** | **1267,53** | **1267,53** | **1267,53** | 24 |
| PDS-10 | 34,36 | **34,29** | 34,96 | 39,00 | 47 |
| PDS-20 | **391,86** | 393,20 | 398,54 | 446,14 | 61 |
| PDS-30 | **342,51** | 344,26 | 349,11 | 387,98 | 74 |
| PDS-40 | **720,50** | 722,78 | 737,06 | 818,32 | 79 |
| PDS-50 | 1483,56 | **1482,55** | 1515,86 | 1571,60 | 78 |
| PDS-60 | **1880,49** | 1885,52 | 1949,57 | 1990,32 | 84 |
| PDS-70 | **2132,50** | 2179,05 | 2179,05 | 2246,25 | 85 |

Tabela 8: Tempo de solução (continuação)

| Problema | TPCLU | TPCLUM | TPCFP | TPCFPM | IT |
|---|---|---|---|---|---|
| PDS-80 | **2508,11** | 2511,95 | 2558,34 | 2645,31 | 83 |
| PDS-90 | 2769,10 | **2768,82** | 2819,85 | 2901,14 | 83 |
| PDS-100 | **3463,63** | 3467,31 | 3540,50 | 3659,54 | 85 |
| QAP12 | **102,83** | 103,07 | 103,53 | 109,72 | 20 |
| NUG07-3rd | 21,46 | 20,69 | 22,85 | **20,44** | 8 |
| NUG08-3rd | 105,95 | 97,53 | 97,74 | **95,82** | 9 |
| NUG12 | 98,03 | 98,18 | **97,04** | 101,33 | 20 |
| NUG15 | 1814,78 | **1792,74** | 1798,99 | 1811,30 | 23 |
| QAP15 | 2902,17 | **1974,65** | 1982,92 | 3010,38 | 23 |
| STE36a | **16212,29** | **16212,29** | **16212,29** | **16212,29** | 37 |
| **Média:** | 1438,03 | **1404,63** | 1419,36 | 1479,65 | 44,75 |

Analisando os dados das Tabela 7 e 8 podemos perceber uma maior variação entre os tempos de processamento, embora o número de iterações do método preditor corretor seja o mesmo independente do método utilizado para detecção das linhas redundantes. O que indica que dependendo do método utilizado e, consequentemente, das linhas identificadas por cada rotina, as iterações do método preditor corretor podem ser tornar mais caras computacionalmente.

Também é possível ter uma noção mais ampla da influência da permutação da base inicial proposta em (Andersen, 1995). Quando utilizamos a forma produto para representação da base e para solucionar os sistemas lineares do algoritmo de detecção de linhas redundantes (Seção 2), o uso dessa permutação diminui significativamente o tempo utilizado pelo método preditor corretor para solucionar os problemas. Para o problema QAP15 essa redução é de cerca de 17 minutos. Por outro lado, quando utilizamos a decomposição LU juntamente com a atualização de Bartels-Golub a não utilização dessa permutação pode reduzir bastante o tempo de processamento. Como por exemplo, para os problemas NUG15 e QAP15 essa redução foi respectivamente de 22,04 segundos e 15,46 minutos.

Novamente a rotina LU detém a maioria dos melhores tempos de processamento, porém a sua média é superior em relação as rotinas LU Modificado e Forma Produto. A rotina LU Modificado é a que apresenta melhor média dos tempos de solução.

Por fim, a Tabela 9, apresenta o tempo total para solucionar os problemas, isto é, o somatório do tempo utilizado no preprocessamento, na detecção de linhas redundantes e pelo método de pontos interiores. As colunas TTLU, TTLUM, TTFP e TTFPM representam, respectivamente, o tempo total (em segundos) quando detectamos as linhas redundantes pelas rotinas LU, LU Modificado, Forma Produto e Forma Produto Modificado.

Através desta tabela podemos verificar que a rotina LU possui a maioria dos melhores tempos totais de solução e a rotina Forma Produto Modificado os piores tempos. Ao compararmos somente as rotinas LU e LU Modificado, embora a rotina LU ainda mantenha a maioria dos melhores tempos de processamento, a diferença de desempenho entre as rotinas é pequena e além disso, a rotina LU Modificado se mostrou bastante superior a LU para os problemas NUG15 e QAP15.

Já ao compararmos as rotinas LU e Forma Produto, a diferença de desempenho também é pequena para maioria dos problemas, exceto para os problemas KEN18, PDS40, PDS50, PDS60, PDS70, PDS80, PDS90 e PDS100 em que a rotina LU é em média 53,82 segundos mais rápida e para os problemas NUG08-3rd, NUG15 e QAP15 em que a rotina Forma Produto é em média 337,07 segundos mais rápida em relação a rotina LU.

Também é possível notar que as rotinas LU Modificado e Forma Produto são as que apresentaram as melhores médias dos tempos totais de solução.

Tabela 9: Tempo total de solução

| Problema | TTLU | TTLUM | TTFP | TTFPM |
|---|---|---|---|---|
| ELS19 | 70,00 | 70,01 | **69,97** | **69,97** |
| CHR25a | 57,57 | 57,57 | **57,53** | **57,53** |
| CHR22b | 27,48 | 27,49 | **27,45** | **27,45** |
| SCR20 | 92,48 | 92,49 | 92,45 | **92,43** |
| CRE-b | **69,04** | 69,62 | 70,35 | 76,14 |
| CRE-d | **36,82** | 37,08 | 37,45 | 37,63 |
| KEN-11 | **11,43** | 11,48 | 11,86 | 12,77 |
| KEN-13 | **112,94** | 114,00 | 117,59 | 118,89 |
| KEN-18 | **1530,98** | 1540,30 | 1589,73 | 1600,75 |
| PDS-06 | 14,44 | 14,63 | **14,34** | 15,34 |
| ROU20 | 1267,78 | 1267,78 | **1267,76** | **1267,76** |
| PDS-10 | 34,92 | **34,89** | 35,53 | 39,63 |
| PDS-20 | **394,17** | 395,61 | 401,01 | 448,93 |
| PDS-30 | **347,47** | 349,43 | 354,43 | 394,23 |
| PDS-40 | **729,55** | 732,32 | 746,90 | 829,88 |
| PDS-50 | 1497,65 | **1497,46** | 1531,43 | 1588,94 |
| PDS-60 | **1901,09** | 1907,44 | 1972,38 | 2015,72 |
| PDS-70 | 2159,94 | **2159,09** | 2209,77 | 2280,41 |
| PDS-80 | **2541,87** | 2549,06 | 2596,95 | 2688,15 |
| PDS-90 | **2809,87** | 2814,06 | 2867,35 | 2953,20 |
| PDS-100 | **3510,63** | 3521,01 | 3597,62 | 3720,22 |
| QAP12 | **103,87** | 104,11 | 103,88 | 110,05 |
| NUG07-3rd | 26,35 | 25,55 | 23,06 | **21,19** |
| NUG08-3rd | 148,50 | 140,57 | 101,48 | **99,30** |
| NUG12 | 101,40 | 101,63 | **97,52** | 101,77 |
| NUG15 | 1840,70 | 1819,54 | **1801,20** | 1813,37 |
| QAP15 | 2909,18 | **1981,45** | 1984,48 | 3011,93 |
| STE36a | 16213,34 | 16213,30 | 16212,31 | **16213,29** |
| **Média:** | 1448,62 | **1416,02** | 1428,40 | 1489,53 |

## 5 Conclusão

O procedimento de detecção de linhas redundantes proposto neste artigo juntamente com a permutação da base inicial proposta em (Andersen, 1995) se mostrou competitivo ao ser comparado com a técnica proposta por (Andersen, 1995). Embora esta técnica detenha a maioria dos melhores tempos de processamento, a diferença de desempenho é pequena, exceto para os problemas destacados na Seção 4. Para os problemas NUG08-3rd, NUG15 e QAP15, o procedimento aqui proposto se mostrou bastante superior na detecção de linhas redundante e no processo de solução. Também é válido destacar que o procedimento apresentado neste trabalho se mostrou como a melhor opção para os problemas em que não há linhas redundantes.

Além disso, a forma produto é uma técnica mais simples de ser implementada em relação a decomposição LU atualizada pelo método de Bartels-Golub.

Outro destaque deste trabalho é que ao não realizar a permutação da base inicial do procedimento proposto em [2], criamos outra abordagem bastante competitiva. Ao compará-lo com a formulação original há uma queda de desempenho se considerarmos apenas a quantidade de melhores tempos de processamento. No entanto, este procedimento se mostrou superior ao original em determinados problemas o que transparece na sua média de tempos de processamento, aliás a melhor média dos tempos de processamento.

Enfim, por meio deste trabalho apresentamos procedimentos eficientes de detecção de linhas redundantes os quais apresentaram resultados promissores ao serem comparados com uma técnica sofisticada de identificação de linhas redundantes.

**Agradecimentos**

Ao CNPq e à FAPESP pelo apoio financeiro.

## Referências

**Adler, I., Resende, M.G.C., Veiga, G. e Karmakar, N.** (1989), An implementation of Karmarkars algorithm for linear programming, *Mathematical Programming*, 44, 297-335.

**Andersen, E. D.** (1995), Finding all linearly dependent rows in large-scale linear programming, *Optimization Methods and Softwares*, 6, 219-227.

**Andersen, E. D. e Andersen, K. D** (1995), Presolving in linear programming, *Mathematical Programming*, 71, 221-245.

**Bocanegra, S., Campos, F. F. e Oliveira, A. R. L.** (2007), Using a hybrid preconditioner for solving large-scale linear systems arising from interior point methods, *Computation Optimization and Applications*, 36, 149-164.

**Burkard, R. S., Karisch, S. e Rendl, A. R. L.** (1991), Qaplib - a quadratic assignment problem library, *European Journal of Operations Research*, 55, 115-119.

**Chvàtal, V.** *Linnear Programming*,W. H. Freeman and Company, New York, 1983.

**Czyzyk, J., Mehrotra, S., Wagner, M. e Wright, S. J.** (1999), PCx an interior point code for linear programming, *Optimization Methods Software*, 11-2, 397-430.

**Dantzig, G. B., Orchard-Hays, W.** (1954), The product form for the inverse in the simplex method, *Mathematical Tables and Others Aids to Computation*, 8 46, 64-67.

**Gondzio, J.** (1996), Multiple centrality corrections in a primal-dual methods for linear programming, *Computation and Optimization and Applications*, 6, 137156.

**Mehrotra, S.** (1992), On implementation of a primal-dual point method, *SIAM Journal on Optimization*, 2, 575601.

**Oliveira, A. R. L. e Sorensen, D. C.** (2005), A new class of preconditioners for large-scale systems from interior point methods for linear programming, *Linear Algebra Its Applications*, 394, 124.

**Suhl, U. H.** (1994), MPOS mathematical optimization system, *European Journal of Operations Research*, 72, 312322.

**Tomlin, J. A. e Welch, J. S.,** (1986), Finding duplicate rows in a linear programming, *Operation Research Letters*, 5, 711.

**Velazco, M. I., Oliveira, A. R. L. e Campos, F. F.** (2010), A note on hybrid preconditioner for solving large-scale normal equations arising from interior point methods, *Optimization Methods and Software*, 25 2 , 321-332.