MÉTODO DUAL SIMPLEX COM A REGRA DE DANTZIG NORMALIZADA

Ricardo Silveira Sousa
Departamento de Educação Matemática
Universidade Federal Fluminense
Santo Antônio de Pádua – Rio de Janeiro - Brasil
rsousa@infes.uff.br

RESUMO

Neste artigo apresentamos a regra de Dantzig normalizada para o método dual simplex com busca unidimensional que é especializado em resolver problemas de otimização linear canalizados (restrições e variáveis canalizadas, chamado formato geral). Experimentos computacionais, em problemas densos e esparsos, mostram que o desempenho desta variante do método simplex é equivalente a regra usual para o método dual simplex com busca unidimensional.

PALAVRAS CHAVE. Otimização linear. Dual simplex. Regra de Dantzig normalizada.

ABSTRACT

In this paper we presented the steepest-edge rule it for the dual simplex method with one-dimensional search that is specialized in solving linear optimization problems lower and upper constrained (i.e., there are lower and upper bounds on constraints and variables, called general format). Computational experiments on dense and sparses problems, show that the performance of this variant of the simplex method is equivalent to the usual rule for the dual simplex method with one-dimensional search.

KEYWORDS. Linear optimization. Simplex dual. Steepest-edge rule.

1. Introdução

O método dual simplex tem atraído considerável interesse, devido à importante aplicação nos métodos de otimização linear inteiro misto Maros (2003) e segundo Bixby (2001), testes computacionais mostram que o desempenho do método dual simplex pode ser superior ao método primal simplex.

Assim, o método dual simplex (Lemke, 1954) foi especializado para o problema primal canalizado:

$$\begin{aligned} &\text{Minimizar} \quad f(\boldsymbol{x}) = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ &\text{Sujeito a: } \boldsymbol{d} \le \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{e}. \end{aligned} \tag{1.1}$$

(ou formato geral, o que constitui uma classe de problemas de otimização linear de grande interesse prático, pois representa vários problemas reais) chamado dual simplex com busca unidimensional (DSBU) cujo desenvolvimento está em Sousa (2005). O problema dual é dado por:

$$\begin{aligned} &\text{Maximizar} \quad h(\boldsymbol{\lambda}) = \sum_{i=1}^{m} h_i(\lambda_i) \\ &\text{Sujeito a:} \quad \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{c}. \end{aligned} \tag{1.2}$$

em que $h_i(\lambda_i) = \begin{cases} e_i\lambda_i & \text{se } \lambda_i \le 0 \\ d_i\lambda_i & \text{se } \lambda_i > 0. \end{cases}$

Experimentos computacionais revelam a importância da busca unidimensional, podendo reduzir o esforço computacional em torno de 25% do critério usual de troca de bases do algoritmo dual simplex. A simplicidade da busca unidimensional faz também com que não implique num custo adicional relevante.

A maioria dos desenvolvimentos nos últimos anos de técnicas e programação nos softwares de otimização linear, é de implementações de métodos tipo dual simplex, pois esses métodos são muitos eficientes, especialmente em resolver problemas de otimização combinatória (Terlaky, 2000 e Bixby 2001).

A vasta literatura de otimização linear contém uma família expressiva de métodos simplex. A flexibilidade inerente de escolher as variáveis que entram e saem da base permite desenvolver vários critérios para a escolha destas variáveis, como por exemplo, a regra de Dantzig normalizada Goldfarb e Reid (1977). Algumas variantes do método dual simplex podem ser encontrados em Dantzig (1963) e Terlaky e Zhang (1993).

Nesse sentido, a regra de Dantzig normalizada foi desenvolvida para tentar reduzir o número de iterações. Ela é conhecida desde os experimentos de Kuhn e Quandt (1963) e Wolfe e Cuttler (1963), mas por causa de sua implementação direta, foi inicialmente rejeitada como impraticável.

No entanto, Goldfarb e Reid (1977) publicaram o primeiro algoritmo do tipo simplex com a regra de Dantzig normalizada praticável. Os autores reportaram testes computacionais preliminares, mostrando a superioridade destra regra em relação a regra usual de Dantzig.

Forrest e Goldfarb (1992) apresentam um estudo mais promissor para esta regra, onde fornecem fórmulas de recorrência para atualização do cálculo das normas para as direções primal e dual simplex. Realizaram vários testes computacionais com problemas da Netlib e concluíram que, em geral, o algoritmo simplex com esta regra realiza menos iterações com relação a regra ususal.

Logo, este trabalho tem como objetivos apresentar a regra de Dantzig normalizada para o DSBU e os experimentos computacionais com esta variante no método dual simplex com busca unidimensional. Na próxima seção tal regra é apresentada e em seguida os resultados. Por último, temos as conclusões, trabalhos futuros e referências bibliográficas.

2.Regra de Dantzig Normalizada

Nos métodos tipo simplex há, em geral, uma flexibilidade de escolha para a troca de base. Por exemplo, no método primal simplex qualquer variável com custo relativo negativo pode ser

escolhida para "entrar na base", ou no método dual simplex, pode-se escolher qualquer restrição primal violada para "deixar a base".

A regra de Dantzig ou regra usual escolhe uma direção simplex que tem o coeficiente mais negativo com o desejo de uma redução significante no valor da função objetivo, no entanto, esta escolha negligência o fato que o comprimento de cada aresta que determina esta direção pode ser diferente. Portanto, a regra de Dantzig normalizada escolhe uma variável que tem o coeficiente normalizado mais negativo, ou seja, fornece o quanto cada aresta contribui a função objetivo por unidade desta direção.

O trabalho de Forrest e Goldfarb (1992) fornece fórmulas recursivas para atualização do cálculo das normas das direções primal e dual simplex. Nos testes realizados com alguns problemas da Netlib, concluíram que, em geral, o algoritmo simplex com esta regra realiza menos iterações em relação a regra usual, além da redução do tempo computacional.

Considerando o método dual simplex clássico (isto é, método dual simplex com a regra de Dantzig), a escolha da variável dual que vai sair da base é definida pela maior violação primal. Assim, $\boldsymbol{\eta}_q = -(\boldsymbol{B}^T)^{-1}\tilde{\boldsymbol{e}}_q$ é a direção dual simplex e a q-ésima restrição é a mais violada. $\boldsymbol{B}$ é a matriz base e $\tilde{\boldsymbol{e}}_q$ é a q-ésima coluna da matriz identidade.

No método dual simplex clássico temos: $x_{B_q} = \min\left\{x_{B_i} < 0 \text{ tal que } i = 1,...,m\right\}$ em que $\boldsymbol{x}_B = \boldsymbol{B}^{-1}\boldsymbol{b}$. No DSBU q é o índice associado ao $\min\left\{d_{N_j} - y_{N_j}, y_{N_j} - e_{N_j}; j = 1,...,m\right\}$, em que $y_{N_j} = \boldsymbol{a}_{N_j}^T\boldsymbol{x}$, $\boldsymbol{x} = \boldsymbol{B}^{-1}\boldsymbol{y}_B$, $y_{B_j} = d_{B_j}$ se $\lambda_{B_j} \geq 0$ e $y_{B_j} = e_{B_j}$ se $\lambda_{B_j} \leq 0$ (veja Sousa, 2005).

Como o custo relativo é dado por: $\nabla h(\hat{\boldsymbol{\lambda}})\boldsymbol{\eta}_q$. ($\nabla h(\hat{\boldsymbol{\lambda}}) = \boldsymbol{b}^T$ é o gradiente da função objetivo dual na solução dual básica $\hat{\boldsymbol{\lambda}}$, o qual é constante no caso clássico, já que $h(\boldsymbol{\lambda}) = \boldsymbol{b}^T\boldsymbol{\lambda}$, diferentemente do DSBU). Portanto, o custo relativo depende da norma da direção simplex e, se quisermos a menor derivada direcional devemos então normalizar a direção simplex e escolher o índice q tal que: $\nabla h(\hat{\boldsymbol{\lambda}})\dfrac{\boldsymbol{\eta}_q}{\|\boldsymbol{\eta}_q\|} = \min\left\{\nabla h(\hat{\boldsymbol{\lambda}})\dfrac{\boldsymbol{\eta}_i}{\|\boldsymbol{\eta}_i\|},\ i = 1,...,m\right\}$.

Porém, se por um lado $\nabla h(\hat{\boldsymbol{\lambda}})\boldsymbol{\eta}_i$ é facilmente calculado pela violação de uma restrição primal, sem a necessidade explícita de $\boldsymbol{\eta}_i$, cujo cálculo envolve a resolução de um sistema linear: $\boldsymbol{\eta}_i = -(\boldsymbol{B}^T)^{-1}\tilde{\boldsymbol{e}}_i$ por outro lado, $\nabla h(\hat{\boldsymbol{\lambda}})\dfrac{\boldsymbol{\eta}_i}{\|\boldsymbol{\eta}_i\|}$ necessita explicitamente de $\boldsymbol{\eta}_i$, tornando a regra de Dantzig normalizada computacionalmente cara (isto inviabilizou as primeiras implementações).

Entretanto, Forrest e Goldfarb (1992) derivam fórmulas recursivas para atualizar as direções primal e dual simplex de uma iteração para a outra.

Considere que a variável de índice q substitui a variável de índice p na base. Sabemos $\bar{\boldsymbol{B}}^{-1} = \boldsymbol{E}^{-1}\boldsymbol{B}^{-1}$, em que $\boldsymbol{B}^{-1}$ é a base atual e $\bar{\boldsymbol{B}}^{-1}$ é a base na iteração subseqüente e $\boldsymbol{E}^{-1}$ é obtida pelo produto de matrizes elementares que correspondem à operação de pivoteamento e que a nova base pode ser escrita em função da base anterior por: $\bar{\boldsymbol{B}} = \boldsymbol{B} + (\boldsymbol{a}_q - \boldsymbol{a}_p)\tilde{\boldsymbol{e}}_p$. Assim, de uma iteração para outra, a inversa da nova base é dada por:

$$\bar{\boldsymbol{B}}^{-1} = \boldsymbol{B}^{-1} - \frac{\boldsymbol{w} - \tilde{\boldsymbol{e}}_p}{\boldsymbol{w}_p}\tilde{\boldsymbol{e}}_p^T\boldsymbol{B}^{-1}, \quad \text{em que } \boldsymbol{w} = \boldsymbol{B}^{-1}\boldsymbol{a}_q. \tag{2.1}$$

Para o desenvolvimento das fórmulas de atualização do DSBU, consideramos o problema de otimização linear no formato geral, onde a matriz $\boldsymbol{A}$ de (1.1) está particionada em

$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{N} \\ \boldsymbol{B} \end{pmatrix}$ em que: $\boldsymbol{B} \in \mathrm{R}^{n \times n}$ é a matriz base dual e $\boldsymbol{N} \in \mathrm{R}^{m \times n}$ é a matriz não base. Note que $\boldsymbol{A} \in \mathrm{R}^{(m+n)\times n}$. No DSBU a matriz base inicial é a matriz identidade.

Logo, a escolha da variável que vai entrar na base, segundo a regra de Dantzig normalizada, é definida pela maior violação primal no limite inferior ou superior (que são os custos relativos para o método dual) dividida pela norma da direção dual simplex, ou seja, escolhe-se um índice não básico que forneça a maior das seguintes razões em valor absoluto: $\frac{d_{N_j} - \hat{y}_{N_j}}{\|\boldsymbol{\eta}_j\|}$ *ou* $\frac{\hat{y}_{N_j} - e_{N_j}}{\|\boldsymbol{\eta}_j\|}$ $j = 1,...,m$. Assim, as direções duais simplex: $\boldsymbol{\eta}_{\mathrm{j}} = \begin{pmatrix} -\left(\boldsymbol{B}^T\right)^{-1} \boldsymbol{a}_{N_j}^T \\ \tilde{\boldsymbol{e}}_{\mathrm{j}} \end{pmatrix}$, precisariam ser calculadas a cada iteração do método para todos os índices não básicos.

Com o propósito de atualizar as direções duais simplex de uma iteração para outra, considere que a variável de índice q ($q \in \{1,..., m\}$) substitui a variável de índice p ($p \in \{1,..., n\}$) na base. Usando (2.1), sabemos que de uma iteração para outra, a inversa da nova base transposta é dada por:

$$(\bar{\boldsymbol{B}}^{-1})^T = (\boldsymbol{B}^{-1})^T - \frac{(\boldsymbol{w} - \tilde{\boldsymbol{e}}_p)}{\boldsymbol{w}_p} \tilde{\boldsymbol{e}}_p^T (\boldsymbol{B}^{-1})^T. \qquad (2.2)$$

em que $\boldsymbol{B}^{-1}$ é a base atual, $\bar{\boldsymbol{B}}^{-1}$ a base na iteração subseqüente e $\boldsymbol{w} = (\boldsymbol{B}^{-1})^T \boldsymbol{a}_q^T$. A identidade acima decorre de: $(\bar{\boldsymbol{B}})^{\mathrm{T}} = (\boldsymbol{B})^{\mathrm{T}} + (\boldsymbol{a}_q - \boldsymbol{a}_p)\tilde{\boldsymbol{e}}_p^T$.

Logo, as novas direções duais simplex são dadas por:

$$\bar{\boldsymbol{\eta}}_{\mathrm{j}} = \begin{pmatrix} -\left(\bar{\boldsymbol{B}}^T\right)^{-1} \boldsymbol{a}_{N_j}^T \\ \tilde{\boldsymbol{e}}_{\mathrm{j}} \end{pmatrix} j = 1,...,m.$$

Para obter as novas normas das direções em função das anteriores, desenvolvemos estas normas ao quadrado, pois facilita a manipulação algébrica.

Portanto, $\left\|\bar{\boldsymbol{\eta}}_{B_j}\right\|^2 = \boldsymbol{a}_{N_J} \bar{\boldsymbol{B}}^{-1} \left(\bar{\boldsymbol{B}}^T\right)^{-1} \boldsymbol{a}_{N_j}^T$. Usando (2.2) temos que:

$$\left\|\bar{\boldsymbol{\eta}}_{B_j}\right\|^2 = \left\|\boldsymbol{\eta}_{B_j}\right\|^2 - \bar{\alpha}_j \boldsymbol{a}_{N_j} \boldsymbol{B}^{-1}(\boldsymbol{w} - \tilde{\boldsymbol{e}}_p) - \bar{\alpha}_j (\boldsymbol{w} - \tilde{\boldsymbol{e}}_p)^T \boldsymbol{B}^{-T} \boldsymbol{a}_{N_J}^T + (\bar{\alpha}_j)^2 (\boldsymbol{w} - \tilde{\boldsymbol{e}}_p)^{\mathrm{T}} (\boldsymbol{w} - \tilde{\boldsymbol{e}}_p).$$

em que $\bar{\alpha}_j = \frac{\alpha_j}{\alpha_q}$, com $\alpha_j = \tilde{\boldsymbol{e}}_p^T \boldsymbol{B}^{-T} \boldsymbol{a}_{N_j}^T$. Pela definição de $\boldsymbol{w}$,

$$\bar{\alpha}_j = \frac{\tilde{\boldsymbol{e}}_p^T \boldsymbol{B}^{-T} \boldsymbol{a}_{N_j}^T}{w_q}.$$

Logo,

$$\left\|\bar{\boldsymbol{\eta}}_{B_j}\right\|^2 = \left\|\boldsymbol{\eta}_{B_j}\right\|^2 - 2\bar{\alpha}_j \boldsymbol{a}_{N_J} \boldsymbol{B}^{-1}(\boldsymbol{w} - \tilde{\boldsymbol{e}}_p) + (\bar{\alpha}_j)^2 (1 + \boldsymbol{w}^T \boldsymbol{w} - 2\tilde{\boldsymbol{e}}_p^T \boldsymbol{w});$$

$$\left\|\bar{\eta}_{B_j}\right\|^2 = \left\|\eta_{B_j}\right\|^2 - 2\bar{\alpha}_j \boldsymbol{a}_{N_J} \boldsymbol{B}^{-1}\boldsymbol{w} + (\bar{\alpha}_j)^2(1+\boldsymbol{w}^T\boldsymbol{w}).$$

Agora, se denotarmos $\boldsymbol{v} = \boldsymbol{B}^{-1}\boldsymbol{w}$ e $\gamma_j = \left\|\eta_j\right\|^2 = 1 + \left\|\eta_{B_j}\right\|^2$, então os quadrados das normas das novas direções duais simplex, são dados por:

$$\bar{\gamma}_j = \gamma_j - 2\bar{\alpha}_j \boldsymbol{a}_{N_J}\boldsymbol{v} + \bar{\alpha}_j\gamma_q \qquad j = 1,\ \ldots,\ m; \quad j \neq q. \tag{2.3}$$

Se $j = q$, temos:

$$\bar{\gamma}_q = \frac{\gamma_q}{\alpha_q^2}. \tag{2.4}$$

Observe que na equação (2.3) existem dois cálculos a fazer, o qual não estaria disponível se estivéssemos executando uma iteração do método simplex com a regra de Dantzig: resolver o sistema $\boldsymbol{v} = \boldsymbol{B}^{-1}\boldsymbol{w}$ e o produto interno das linhas não básicas da matriz $\boldsymbol{A}$ com o vetor $\boldsymbol{v}$. Há necessidade também do cálculo inicial dos quadrados das normas das direções.

Tendo também como ponto de partida a identidade (2.2), de uma maneira similar esta técnica pode ser aplicada no cálculo das novas direções primal simplex.

Forrest e Goldfarb (1992) trabalharam com 20 problemas da Netlib, o que na média tinham 3412 linhas e 12492 colunas. Os autores utilizaram o método primal e dual, sendo que este último teve o pior desempenho. A média de iterações para o algoritmo de Dantzig foi 6,3*m* enquanto o algoritmo com a regra de Dantzig normalizada realizou 1,9*m* iterações, em que *m* é o número de restrições.

Motivado pelo trabalho de Forrest e Goldfarb (1992) e no bom desempenho do método dual simplex com busca unidimensional comparado ao pacote CPLEX, relativo ao número de iterações (Sousa, 2005 e Silva *et al*., 2007), implementamos a regra de Dantzig normalizada para o DSBU, para resolver problemas gerados aleatoriamente de médio porte e alguns da Netlib. Os resultados computacionais se encontram na próxima seção,

3. Experimentos Computacionais

Daremos os resultados computacionais obtidos com o método dual simplex com busca unidimensional, para as regras de Dantzig e Dantzig normalizada, onde são feitas algumas comparações para problemas gerados aleatoriamente com dados densos e esparsos, além de cinco problemas da Netlib.

O DSBU foi implementado utilizando a forma produto da inversa. Para cada tamanho (dimensão) do problema, que define um exemplar com *m* restrições e *n* variáveis (veja Figura 1), foram rodados 20 exemplos. Os testes foram realizados em um computador de 1.6 GHz com 2.0 GB de RAM. Os resultados obtidos com DSBU foram comparados também com o CPLEX na versão 12.1 utilizando o algoritmo dual simplex.

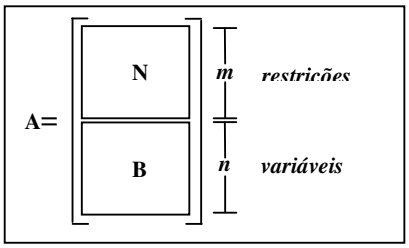

Figura 1 - Representação da matriz na forma geral (problema 1.1)

O tempo (em segundos) e o número de iterações, reportados a seguir, são as médias dos 20 exemplos para cada tamanho do problema. Para a apresentação dos resultados, considere que **CP** indica que o procedimento *presolve* do CPLEX foi utilizado e **SP** que não foi usado, **D** representa a regra de Dantzig, **DN** a regra de Dantzig normalizada, *m* o número de restrições e *n* o número de variáveis. Ressaltamos que o *presolve* visa reduzir o tamanho do problema original.

A Tabela 3.1 apresenta os resultados para as duas regras, com 100% dos elementos das restrições diferentes de zero.

Tabela 3.1 - DSBU ×CPLEX – Problemas Densos

Exemplar		DSBU				CPLEX			
		Iterações		Tempo		Iterações		Tempo	
	m*×*n	**D**	**DN**	**D**	**DN**	**CP**	**SP**	**CP**	**SP**
1	***10*×*400***	*33,7*	*34,8*	*0,09*	*0,12*	*31,0*	*32,1*	*0,03*	*0,02*
2	***20*×*400***	*61,8*	*63,5*	*0,13*	*0,29*	*60,5*	*58,6*	*0,03*	*0,02*
3	***100*×*100***	*131,4*	*129,7*	*0,03*	*0,05*	*187,8*	*199,7*	*0,12*	*0,08*
4	***200*×*200***	*292,1*	*300,8*	*0,28*	*0,53*	*484,2*	*521,0*	*0,71*	*0,70*
5	***200*×*400***	*382,6*	*375,3*	*1,33*	*1,97*	*666,3*	*694,2*	*1,48*	*1,34*
6	***400*×*400***	*795,3*	*725,7*	*3,20*	*5,53*	*1327,8*	*1408,1*	*6,35*	*6,65*
7	***500*×*500***	*838,8*	*907,2*	*5,86*	*11,40*	*1826,4*	*2065,6*	*12,5*	*14,02*
8	***1000*×*1000***	*1820,6*	*1788,3*	*48,6*	*91,6*	*5155,3*	*5652,3*	*112,3*	*124,4*
9	***500*×*4000***	*2333,5*	*2425,5*	*492,3*	*1350,2*	*3508,3*	*3810,5*	*69,5*	*73,11*
10	***500*×*5000***	*2689,2*	*2763,1*	*888,5*	*3570,3*	*3977,0*	*4091,7*	*99,2*	*88,3*

Observamos, com relação ao DSBU, que apenas para os exemplares 3, 5, 6 e 8 a regra de Dantzig normalizada foi superior a regra usual no número de iterações, mas significante somente no exemplar 6, enquanto o tempo computacional foi muito favorável à regra de Dantzig, em especial para os problemas maiores, chegando a ser quase 4 vezes mais rápida no exemplar 10.

Notamos ainda que o DSBU usual e com a regra DN foram muito melhores que o CPLEX com ou sem a utilização do *presolve* tanto em iterações quanto em tempo, envolvendo os exemplares de 1 a 8. Para os exemplos maiores 9 e 10, o tempo do CPLEX foi muito menor do que o DSBU, ao passo que realizou muito mais iterações, chegando a fazer cerca de 50% mais esforço.

Trabalhamos também com uma classe esparsa de problemas, para os quais a matriz de restrição é da forma escada (Veja Figura 2) e os elementos tem valores positivos e negativos na casa de unidades.

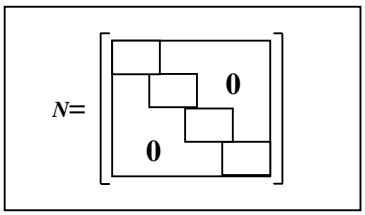

Figura 2 - Representação da matriz ***N*** não básica na forma escada

Na maioria, os problemas foram gerados de tal forma que a esparsidade ficasse abaixo de 1%, com exceção de um exemplar, pois muitos problemas reais têm esta característica.

Os elementos não nulos da matriz ***N*** estão confinados nos blocos e foram gerados aleatoriamente. Considere nas tabelas a seguir que, *nb* representa o número de blocos, *m*1 e *n*1 representam os números de linhas e colunas, respectivamente em cada bloco e *cc* o número de colunas em comum a cada dois blocos consecutivos.

Para o gerador utilizado, 80% dos elementos do vetor custo foram iguais a zero, de tal forma que várias soluções básicas duais sejam degeneradas. O que gera uma dificuldade maior para o método de resolução. Portanto, utilizamos o procedimento de expansão (Gill *et al.*, 1989) para tratar degeneração no DSBU que obteve bons resultados sem acréscimo significativo no tempo computacional (Sousa, 2010).

O número de iterações, reportados a seguir, são as médias dos resultados da resolução dos 20 exemplares para cada tamanho (dimensão) do problema, sendo que % representa a porcentagem de elementos diferente de zero em relação ao total na matriz ***N***.

As Tabelas 3.2 e 3.3 apresentam o número de iterações e o tempo computacional respectivamente, para os primeiros problemas esparsos, confrontando o DSBU com o CPLEX.

Tabela 3.2 – Iterações: DSBU × CPLEX – Problemas Esparsos I

Problemas						***DSBU***		***CPLEX***	
nb	***m1***	***n1***	***cc***	***%***	***m×n***	***D***	***DN***	***CP***	***SP***
20	*10*	*40*	*0*	*5,00*	***200×800***	*616,3*	*602,3*	*300,6*	*323,6*
400	*3*	*4*	*1*	*0,33*	***1200×1201***	*948,3*	*949,5*	*1046,4*	*1257,5*
500	*1*	*2*	*0*	*0,20*	***500×1000***	*387,3*	*387,3*	*9,1*	*459,6*
500	*3*	*5*	*2*	*0,20*	***1500×1502***	*1328,0*	*1330,6*	*1502,3*	*1714,3*
900	*2*	*2*	*0*	*0,11*	***1800×1800***	*1236,2*	*1236,2*	*720,5*	*1350,8*

Notamos que a regra de Dantzig normalizada obteve um desempenho muito similar a regra usual, e ambas foram superiores ao CPLEX nos exemplares 2, 3, 4 e 5 sendo que no terceiro, foi sem o uso do *presolve*. Neste caso tal procedimento se mostrou muito eficiente, fazendo apenas 2% do número de iterações de SP. Observe ainda que neste exemplar 3 as variáveis são independentes (pois *cc*=0), $n > m$ e os blocos são muito pequenos (1×2). Embora o exemplar 1 tenha *cc*=0, o *presolve* não teve o mesmo desempenho, o que deve ser ocasionado pelo tamanho de cada bloco com 10 restrições e 40 variáveis.

Este é um procedimento que não temos em nossa implementação, mas que deverá ser alvo de pesquisa futura.

Tabela 3.3 – Tempo: DSBU × CPLEX – Problemas Esparsos I

Problemas						***DSBU***		***CPLEX***	
nb	***m1***	***n1***	***cc***	***%***	***m×n***	***D***	***DN***	***CP***	***SP***
20	*10*	*40*	*0*	*5,00*	***200×800***	*7,2*	*18,6*	*0,05*	*0,04*
400	*3*	*4*	*1*	*033*	***1200×1201***	*46,4*	*97,4*	*0,10*	*0,09*
500	*1*	*2*	*0*	*0,20*	***500×1000***	*9,1*	*19,9*	*0,01*	*0,03*
500	*3*	*5*	*2*	*0,20*	***1500×1502***	*104,3*	*269,9*	*0,12*	*0,09*
900	*2*	*2*	*0*	*0,11*	***1800×1800***	*119,8*	*242,7*	*0,07*	*0,08*

No que se refere aos tempos gastos, a tabela acima mostra uma superioridade muito grande do pacote CPLEX, e que o tempo do método com DN foi na média um pouco mais do que o dobro do método com D, o que é uma grande desvantagem, assim acomo foi observado nos problemas densos. Mas, que pode ser melhorado.

A seguir, as Tabelas 3.4 e 3.5 apresentam os resultados para problemas maiores, reportando o número de iterações e os tempos respectivamente.

Tabela 3.4 – Iterações: DSBU × CPLEX – Problemas Esparsos II

problemas						***DSBU***		***CPLEX***	
nb	***m1***	***n1***	***cc***	***%***	***m×n***	***D***	***DN***	***CP***	***SP***
500	*2*	*4*	*0*	*0,20*	***1000×2000***	*877,0*	*877,0*	*943,1*	*1154,2*
500	*2*	*6*	*0*	*0,20*	***1000×3000***	*979,5*	*979,5*	*1082,1*	*1185,2*
700	*2*	*2*	*0*	*0,14*	***1400×1400***	*954,3*	*948,2*	*596,1*	*1072,9*
1000	*1*	*3*	*1*	*0,15*	***1000×2001***	*1,6*	*1,6*	*0,5*	*1,6*
1000	*2*	*4*	*0*	*0,10*	***2000×4000***	*1742,6*	*1742,6*	*1759,3*	*2002,4*
1000	*3*	*5*	*2*	*0,17*	***3000×3002***	*2650,4*	*2648,7*	*3183,5*	*340,3*
1000	*2*	*6*	*1*	*0,12*	***2000×5001***	*3,4*	*3,4*	*4,3*	*4,4*
2000	*2*	*4*	*1*	*0,07*	***4000×6001***	*5,3*	*5,0*	*4,6*	*6,3*
3000	*2*	*4*	*1*	*0,04*	***6000×9001***	*5,6*	*5,6*	*4,6*	*6,3*
4000	*2*	*4*	*1*	*0,03*	***8000×12001***	*6,2*	*7,6*	*4,5*	*8,3*

Os resultados mostram novamente o desempenho muito similar entre as duas regras para o DSBU, com muitos empates. Ambas foram melhores que o pacote CPLEX sem o uso do *presolve* e em algumas situações com o uso desta ferramenta, como no caso dos exemplares 1, 2, 5, 6, 7. O que mostra um excelente desempenho do DSBU.

Um fato muito curioso, é o baixíssimo número de iterações (tanto para o DSBU quanto para o CPLEX) para os problemas maiores e para o exemplar 4 também. O que se tem em comum nesses casos é: cc=1 e *n* é maior do que *m*. Mas a Tabela 3.2 apresenta um exemplar em que *cc*=1 com muitas iterações, no entanto, o número de restrições e variáveis são praticamente iguais. Diante dessa situação, resolvemos investigar um pouco mais problemas com estas características, cujo os resultados estão nas Tabelas 3.6 e 3.7. Mas antes os tempos dos problemas acima são relatados Tabela 3.5.

Tabela 3.5 – Tempo: DSBU × CPLEX – Problemas Esparsos II

problemas						***DSBU***		***CPLEX***	
nb	***m1***	***n1***	***cc***	***%***	***m×n***	***D***	***DN***	***CP***	***SP***
500	*2*	*4*	*0*	*0,20*	***1000×2000***	*82,4*	*163,4*	*0,05*	*0,03*
500	*2*	*6*	*0*	*0,20*	***1000×3000***	*183,3*	*410,1*	*0,08*	*0,06*
700	*2*	*2*	*0*	*0,14*	***1400×1400***	*53,4*	*132,8*	*0,03*	*0,03*
1000	*1*	*3*	*1*	*0,15*	***1000×2001***	*0,15*	*0,35*	*0,00*	*0,00*
1000	*2*	*4*	*0*	*0,10*	***2000×4000***	*783,4*	*3402,3*	*0,12*	*0,11*
1000	*3*	*5*	*2*	*0,17*	***3000×3002***	*873,6*	*2090,5*	*0,30*	*0,21*
1000	*2*	*6*	*1*	*0,12*	***2000×5001***	*3,2*	*4,8*	*0,02*	*0,01*
2000	*2*	*4*	*1*	*0,07*	***4000×6001***	*10,2*	*26,8*	*0,02*	*0,03*
3000	*2*	*4*	*1*	*0,04*	***6000×9001***	*38,7*	*95,7*	*0,02*	*0,02*
4000	*2*	*4*	*1*	*0,03*	***8000×12001***	*108,7*	*270,7*	*0,13*	*0,06*

Em relação ao tempo, mais uma vez podemos observar a clara superioridade do CPLEX frente do DSBU. Para os exemplares 1000×2001 e 2000×5001 a diferença foi menor em relação aos demais, pois o DSBU realizou pouquíssimas iterações (veja Tabela 3.4) e o tamanho destes problemas não está entre os três maiores.

Tabela 3.6 – Iterações: DSBU × CPLEX – Problemas Esparsos III

Problemas						DSBU		CPLEX	
nb	**m1**	**n1**	**cc**	**%**	**m×n**	**D**	**DN**	**CP**	**SP**
700	1	2	1	0,29	**700×701**	473,3	473,3	277,4	526,7
500	3	4	1	0,27	**1500×1501**	1190,3	1189,0	1384,2	1557,8
1500	1	2	1	0,13	**1500×1501**	1035,7	1035,7	580,3	1140,0
400	4	5	1	0,31	**1600×1601**	1355,0	1353,3	1523,7	1715,2
1000	2	5	1	0,12	**2000×4001**	4,6	4,6	2,6	4,6
2000	1	3	1	0,07	**2000×4001**	1,5	1,5	0,7	1,5
2500	1	4	1	0,05	**2500×7501**	1,7	1,7	1,0	1,7
1500	2	6	1	0,08	**3000×7501**	4,3	4,0	4,0	6,1
1000	2	5	2	0,17	**2000×3002**	7,1	7,1	5,0	7,4
2500	1	4	2	0,08	**2500×5002**	3,3	3,3	1,0	3,7
2000	1	5	2	0,08	**2000×6002**	2,4	2,4	1,5	2,4
2000	1	5	3	0,12	**2000×4003**	4,6	4,6	2,5	4,3
1500	2	6	3	0,13	**3000×4503**	8,3	8,3	10.0	12,5

Os resultados acima confirmam que, mesmo variando um pouco o número de colunas em comum e as dimensões de cada blocos, quando $n > m$, o número de iterações fica na casa de unidades para resolver problemas de médio porte. O que caracteriza um excelente desempenho para os algoritmos utilizados nos testes, envolvendo o gerador utilizado. Isto indica que problemas com esta característica, são mesmos solucionados com pouquíssimo esforço do método simplex. A seguir, são relatados os tempos.

Tabela 3.7 – Tempo: DSBU × CPLEX – Problemas Esparsos III

Problemas						DSBU		CPLEX	
nb	**m1**	**n1**	**cc**	**%**	**m×n**	**D**	**DN**	**CP**	**SP**
700	1	2	1	0,29	**700×701**	7,4	15,1	0,03	0,01
500	3	4	1	0,27	**1500×1501**	119,9	189,2	0,12	0,07
1500	1	2	1	0,13	**1500×1501**	71,2	152,5	0,04	0,04
400	4	5	1	0,31	**1600×1601**	102,3	259,7	0,13	0,08
1000	2	5	1	0,12	**2000×4001**	2,9	8,9	0,01	0,01
2000	1	3	1	0,07	**2000×4001**	1,2	3,1	0,01	0,01
2500	1	4	1	0,05	**2500×7501**	5,9	17,4	0,01	0,01
1500	2	6	1	0,08	**3000×7501**	13,5	37,5	0,02	0,02
1000	2	5	2	0,17	**2000×3002**	2,1	12,6	0,01	0,02
2500	1	4	2	0,08	**2500×5002**	3,7	7,5	0,01	0,01
2000	1	5	2	0,08	**2000×6002**	4,1	10,8	0,01	0,01
2000	1	5	3	0,12	**2000×4003**	2,5	11,7	0,01	0,01
1500	2	6	3	0,13	**3000×4503**	7,3	38,6	0,04	0,02

Novamente, observamos da tabela acima a enorme vantagem do CPLEX no tempo computacional e a desvantagem de DN em relação a D. Vale a pena observar que o pacote CPLEX explora eficientemente a esparsidade dos problemas e no caso de uso do *presolve*, na maioria dos problemas, a estrutura da matriz deve ser bem aproveitada pelo software antes de resolvê-los.

De um modo geral notamos também que o tempo gasto com *presolve* é maior, dado o trabalho que precisa ser feito, como por exemplo, identificar e remover restrições reduntantes e fixar variáveis.

Apresentamos a seguir, o número de iterações para resolver 5 problemas da Netlib, os quais são pequenos, mas difíceis de serem resolvidos, no sentido que são muitos sensíveis a qualquer perturbação.

Tabela 3.8 –Iterações: DSBU ×CPLEX – Problemas da Netlib

problemas			***DSBU***		***CPLEX***	
nome	***%***	***m×n***	**D**	**DN**	**CP**	**SP**
afiro	*8.30*	***25×39***	*15*	*16*	*6*	*11*
adlitlle	*2.49*	***53×196***	*108*	*138*	*84*	*89*
kb2	*12.79*	***43×52***	*31*	*144*	*27*	*19*
Sc50a	*5.52*	***49×48***	*55*	*48*	*12*	*49*
Sc50b	*5.12*	***48×48***	*50*	*50*	*20*	*54*

Notamos, com relação ao DSBU, que DN teve o pior desempenho, mas empatando em um problema e ganhando em outro. Observa-se que o pacote CPLEX foi na média muito superior ao DSBU com exceção do *kb2*. Um fato interessante neste problema foi que, o *presolve* piorou o número de iterações.

Não relatamos o tempo aqui, pois teve o mesmo padrão observado para os outros problemas que tratamos anteriormente e foi nulo (0,00 segundos).

No geral, o melhor desempenho do CPLEX para estes problemas muitos degenerados e sensíveis, se deve ao uso de ferramentas para combater problemas desta natureza. Ele perturba os dados do problema original. Uma vez resolvido, o solver remove a perturbação, redefinindo dados do problema aos seus valores originais.

Embora o DSBU esteja usando apenas um procedimento para tratar degeneração, seu resultado foi muito melhor quando não se utiliza o método de expansão, conforme foi observado por Sousa (2010).

Ressaltamos que em muitos casos, em especial para os problemas com baixo número de iterações do CPLEX o *presolve* removeu cerca de 95% das linhas e das colunas do problema original, algo que facilita muito a resolucao do novo problema. Mas, em alguns exemplares essa ferramenta não se mostrou muito eficiente.

4. Conclusões e Trabalhos Futuros

Comparando os resultados, fica clara a superioridade do método DSBU frente ao CPLEX quando não se faz o uso do *presolve*, embora o tempo não seja favorável. E em alguns casos, mesmo com o uso desta ferramenta, foi exigido menos eforço do nosso algoritmo. Um resultado relevante.

Os experimentos computacionais revelam ainda um desempenho equivalente entre as regras tratadas, evidenciando-se assim, duas classes de problemas (densa e esparsa) em que regra Dantzig normalizada não tem uma superioridade clara, conforme já foi observado nos testes preliminares de Sousa (2003). Por outro lado, diferentemente do que foi relatado por Forrest e Goldfarb (1992) em problemas da Netlib.

E mais, notamos com estes primeiros testes, o excelente desempenho (pouquíssimas iterações) dos algoritmos do tipo dual simplex (DSBU e CPLEX) para resolver problemas de médio porte em que n é maior do que m. Situação observada também, algumas vezes, quando se utiliza algum método de pontos interiores, que perfaz 10 a 40 iterações na resolução de alguns problemas de otimização linear (Roos e Terlaky, 1996).

Contudo, deverá ser objeto de investigação, a melhora no tempo do DSBU com a regra de Dantzig normalizada e a utilização de outro gerador de problemas, que envolva a disposição dos dados gerados e seus valores, para simular, por exemplo, alguns problemas oriundos da programação da produção.

Agradecimentos
Este trabalho teve o apoio da Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação da Universidade Federal Fluminense-PROPP/UFF e da Fundação de Apoio à Pesquisa do Estado do Rio de Janeiro-FAPERJ.

5. Referências Bibliográficas

Bixby, R. E., Solving Real-World Linear Programs: A Decade and More of Progress, *ILOG*, Inc and Rice University. http://www.caam.rice.edu/~bixby/default.htm, 2001.

Dantizg, G. B. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

Dickson, J. C. and Frederick, F. P. (1960) "A Decision Rule for Improvement Efficiency in Solving Linear Programming Problems with the Simplex Algorithm" *Comm. ACM* 3, 509-512.

Forrest, J. J. and Goldfarb, D. (1992) "Steepest-Edge Simplex Algorithms for Linear Programming" *Mathematical Programming* 57, 341-374.

Gill, P. E., Murray, W., Saunders, M.A., Wright, M. H. (1989) "A Practical Anti-cycling Procedure for Linearly Constrained Optimization" *Math. Programming,* 45, 437-474.

Goldfarb, D. and Reid, J. K. (1977) "A Practicable Steepest-Edge Simplex Algorithm" *Mathematical Programming*, 12, 361-371.

Klee, V. and Minty, G. J. (1972) "How Good is the Simplex Algorithm?" in *Shish, Proc. 3rd Sympos. Inequalities*, AP, New York, 159-175.

Kuhn, H. W. and Quandt, R. E. (1963) "An Experimental Study of the Simplex Method" in Metropolis *et al.* (Eds.), Proc. *15th Sympos. Appl. Math*., Amer. Math. Soc., Providence, R.I., 107-124.

Lemke, C. E. (1954) The Dual Method of Solving the Linear Programming Problem, *Naval Research Logistics Quarterly*, 1, 36-47.

Maros, I., *Computational Techniques of the Simplex Method*, Kluwer Academic Publishers, USA, 2003.

Roos, C. and Terlaky, T. (1996) "*Advances in Linear Optimization*" Report Nr. 96-118, Faculty of Technical Mathematics and Informatics, Delf, Netherland.

Silva, C. T., Sousa, R. S., e Arenales, M. N. (2007) Métodos tipo dual simplex para problemas de otimização linear canalizados e esparsos. *Pesquisa Operacional*, v. 27, p. 457-486.

Sousa, R. S. (2010) Método Dual Simplex com Busca Unidimensional: degeneração e ciclagem, In: Simpósio Brasileiro de Pesquisa Operacional, 2010, Bento Gonçalves.

Sousa, R. S. (2005) *Métodos Tipo Dual Simplex para Problemas de Otimização Linear Canalizados*, Tese de Doutorado, ICMC-USP.

Sousa, R. S. e Arenales, M, N. Método Dual Simplex com a Regra de Steepest-Edge para Resolver Problemas Lineares Canalizados. In: XXXV SBPO Simpósio Brasileiro de Pesquisa Operacional, 2003, Natal.

Terlaky, T. (2000) "*Linear Programming, Simplex-Type Algorithms*" Department of Computing and Software, McMaster University, Hamilton, ON, Canada. http://www.cas.mcmaster.ca/~terlaky.

Terlaky, T. and Zhang, S. (1993) "Pivot Rules for Linear Programming: A Survey on Recent Theorical Developments" *Annals of Operation Research* 46, 203-233.

Wolfe, P. and Cutler, L. (1963) "Experiments in Linear Programming" in Graves and Wolfe (Eds.) *Recent Advances in Mathematical Programming*, McGraw Hill, New York, 177-200.