MÉTODO DUAL SIMPLEX COM BUSCA UNIDIMENSIONAL: DEGENGARAÇÃO E CICLAGEM

Ricardo Silveira Sousa
Departamento de Ciência da Computação
Universidade Federal de Lavras
Campus Universitário Cx. Postal 3037
Lavras – Minas Gerais - Brasil CEP 37200-000
rsousa@dcc.ufla.br

RESUMO

Neste artigo apresentamos o método de expansão para tratar degeneração no método simplex e o adaptamos 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 mostram que é necessário utilizar algum procedimento anti-degeneração para resolver problemas degenerados, em especial, para alguns problemas da Netlib.

PALAVRAS CHAVE. Otimização linear. Método dual simplex. Degeneração.

ABSTRACT

In this paper we presented expand method to treat degeneracy in the simplex method and we adapted 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 shows that it is necessary to use some procedure anti-degeneracy to solve degenerate problems, especially, for some problems of Netlib.

KEYWORDS. Linear optimization. Simplex dual method. Degeneracy.

1. Introdução

O problema primal de otimização linear com restrições canalizadas (ou, formato geral, conforme Vanderbei (1997) é definido por:

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

em que $\boldsymbol{A} \in \mathrm{R}^{m \times n}$; $\boldsymbol{d}, \boldsymbol{e} \in \mathrm{R}^m$ com $d_i \le e_i \;\; i=1,...,m$; $\boldsymbol{c}, \boldsymbol{x} \in \mathrm{R}^n$.

A classe de problemas de otimização linear com restrições canalizadas é de grande interesse prático, pois representa vários problemas reais, tais como problemas de mistura, planejamento, etc., cujas restrições, ou parte delas, são definidas por limitantes tanto superiores como inferiores, decorrentes de tolerâncias de especificações técnicas, demanda, etc.

O problema dual de (1.1) desenvolvido em Sousa (2005) 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 $h_i(\lambda_i) = \begin{cases} e_i\lambda_i & \text{se } \lambda_i \le 0 \\ d_i\lambda_i & \text{se } \lambda_i \ge 0. \end{cases}$

Observe que o problema dual é um problema de otimização linear onde a função objetivo é côncava linear por partes. Então, o método dual simplex (Lemke, 1954) foi especializado para o problema canalizado chamado dual simplex com busca unidimensional (DSBU) cujo desenvolvimento está em Sousa (2005).

Segundo Maros (2003), o método dual simplex tem atraído considerável interesse, devido à importante aplicação nos métodos de otimização linear inteiro misto, os quais resolvem uma sequência de problemas de otimização linear, com característica de que uma solução básica dual factível de boa qualidade é sempre disponível para o problema seguinte da sequência. Segundo Bixby (2001), testes computacionais mostram que o desempenho do método dual simplex pode ser superior ao método primal simplex.

É importante observar que a evolução das implementações computacionais dos resolvedores lineares teve um papel fundamental no progresso da Otimização Linear (Sousa, 2005 e Silva *et al*.,(2007)). Um exemplo disto é o pacote CPLEX, que vem sendo melhorado desde sua primeira versão lançada em 1988.

Experimentos computacionais revelam a importância da busca unidimensional, podendo reduzir o esforço computacional a 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 ao método simplex.

Em problemas altamente degenerados, isto é, muitas soluções básicas com variáveis em seus limitantes, tendem a provocar no método tipo simplex o fenômeno de estagnação, que é a mudança de base sem melhorar o valor da função objetivo.

Podemos observar tal fenômeno quando, usamos o método dual simplex com busca unidimensional em alguns problemas da Netlib. A princípio, pensávamos que a busca unidimensional fosse capaz de evitar ou, pelo menos, reduzir a estagnação, mas observamos que a estagnação persistiu nesses problemas. Isto nos motivou a rever a literatura sobre estagnação e adaptá-la ao método dual simplex com busca unidimensional.

Sabemos que o maior esforço computacional realizado pelo método dual simplex com busca unidimensional, consiste na resolução de três sistemas lineares em cada iteração, em que métodos iterativos podem ser usados para resolver com sucesso tais sistemas, como foi observado por Sousa (2005) e Sousa *et al*., (2006).

Mas, neste trabalho estamos preocupados em resolver os problemas que são altamente degenerados como muitos problemas da Netlib que são usados para avaliar o desempenho de novos algoritmos do tipo simplex.

De acordo com (Zornig, 2008) problemas que envolvem degeneração e ciclagem no método simplex têm atraído interesse nos últimos anos como podemos citar os trabalhos de Zornig (2006 e 2008), Pan (1998), Raymond *et al*., (2010) e Elhallaoui *et al*., (2008).

Portanto, este trabalho tem como objetivo principal apresentar o método de expansão (Gill *et al*., 1989) para tratar degeneração e os experimentos computacionais para o método dual simplex com busca unidimensional com este procedimento anti-degeneração adaptado.

2. Degeneração e Ciclagem

Os métodos do tipo simplex percorrem soluções básicas, digamos, $\boldsymbol{B}^0$, $\boldsymbol{B}^1$, $\boldsymbol{B}^2$, ... , $\boldsymbol{B}^t$, ... ($\boldsymbol{B}^t$: base na iteração t), de modo que: $f(\boldsymbol{x}^0) \geq f(\boldsymbol{x}^1) \geq f(\boldsymbol{x}^2)$ ... $\geq f(\boldsymbol{x}^t) \geq$ ...em que $\boldsymbol{x}^t$ é a solução básica associada à base $\boldsymbol{B}^t$ (no caso dual: $h(\boldsymbol{\pi}^0) \leq h(\boldsymbol{\pi}^1) \leq$... $h(\boldsymbol{\pi}^t) \leq$... em que h é a função objetivo dual e $\boldsymbol{\pi}^t$ é a solução básica dual associada à base $\boldsymbol{B}^t$). Se as soluções básicas são não-degeneradas então: $f(\boldsymbol{x}^0) > f(\boldsymbol{x}^1)$ ... $> f(\boldsymbol{x}^t) >$... de modo que as soluções básicas são distintas e, portanto, nunca uma base é repetida. Entretanto, se soluções degeneradas ocorrem (as quais podem ser representadas por partições básicas distintas), pode-se ter um seqüência de soluções básicas associadas as partições básicas distintas tal que:... $f(\boldsymbol{x}^t) < f(\boldsymbol{x}^{t+1}) = f(\boldsymbol{x}^{t+2}) = \ldots = f(\boldsymbol{x}^{t+k}) < f(\boldsymbol{x}^{t+k+1})$.

Dizemos que há uma "*estagnação*" do método (*stalling*) em k iterações (um mesmo vértice é representado por distintas partições básicas, que ocorre somente se a solução for degenerada), quando após k-mudanças de base o valor da função objetivo permanece o mesmo.

Entretanto, poder-se-ia ocorrer que $f(\boldsymbol{x}^{t+1}) = \ldots = f(\boldsymbol{x}^{t+k})$ e as partições básicas associadas à $\boldsymbol{x}^{t+1}$ e $\boldsymbol{x}^{t+k}$ serem as mesmas, de modo que o ciclo se repetiria infinitamente. Este fenômeno é chamado ciclagem.

Embora degeneração abra a possibilidade de ocorrer ciclagem ela dificilmente ocorre na prática (Maros, 2003). Este problema foi reconhecido e demonstrado por Beale (1955). Várias ferramentas foram desenvolvidas para evitar ciclagem, as duas mais conhecidas são o método simplex lexicográfico (Charnes, 1952) que é equivalente) à técnica de perturbação de Dantzig *et al*. (1955) segundo Terlaky (2000) e o método mais popular para evitar ciclagem é conhecido pela regra de Bland (1977) que foi discutida em Sousa (2000).

Para a escolha da variável que entra na base a regra de Bland seleciona o menor índice entre as variáveis não básicas com custo reduzido negativo. Se não existe tal variável a solução atual é ótima. Para a escolha da variável que sai, o menor índice é escolhido tal que após a mudança, a base permaneça factível e o índice desta variável seja o menor entre todos aqueles com tal possibilidade.

Bland demonstrou que o método simplex é finito com esta regra. Entretanto, o desempenho computacional do método simplex, sob esta regra, decai (Avis e Chvátal, 1978).

O trabalho de Gass e Vinjamuri (2004) apresenta uma coleção de problemas que produzem ciclagem quando resolvidos manualmente pelo método simplex com os dados fracionários e, por curiosidade, aplicamos o método dual simplex com busca unidimensional para estes problemas, cujos os resultados serão apresentados na seção 4.

Segundo Gass e Vinjamuri, existem algumas condições necessárias para se construir um exemplo capaz de produzir o fenômeno de ciclagem no método simplex.

Assim, considere um problema de otimização linear na forma padrão com m restrições e n variáveis, com empate na escolha da variável que entra ou sai da base (pela regra usual de Dantzig) em que a escolha se faz pelo menor índice das candidatas. Com essa regra, para ciclagem ocorrer deve-se ter $m \geq 2$, $n \geq m+3$, e $n \geq 6$. E para ciclagem ocorrer em um vértice que não seja a solução ótima, deve-se ter $m \geq 3$, $n \geq m+3$ e $n \geq 7$.

Assim, um exemplo com somente duas variáveis não pode ciclar; um exemplo de ciclagem deve ter pelo menos 6 variáveis, o mínimo de duas restrições de igualdade e pelo menos três variáveis não básicas. Uma solução básica factível com degeneração simples (somente uma variável básica é zero) não pode ciclar (Gass, 1993) e, o 'comprimento' mínimo de um ciclo é de 6 iterações (Yudin e Golshtein, 1965). Todos os exemplares podem ser consultados em Sousa (2005).

O primeiro exemplo de um problema em que o método simplex cicla foi construído por Hoffman (1953) que tem sido estudado na literatura. Zornig (2008) apresenta condições necessárias e suficientes, em uma estrutura de permutação, para que ocorra ciclagem em alguns exemplares tendo como base esse exemplo de Hoffman.

Apesar da ciclagem ser um problema importante no método simplex, ela dificilmente ocorre na prática, diferentemente da degeneração que precisa ser tratada em algoritmos do tipo simplex. Logo, apresentamos na próxima seção o método de expansão para evitar o fenômeno de estagnação para o qual se bons resultados práticos (Maros, 2003).

3.Método de Expansão

Nesta seção apresentamos uma técnica desenvolvida por Gill *et al*. (1989) para tratar a estagnação no método simplex. Os autores realizaram experimentos computacionais com 53 problemas da Netlib e obtiveram uma melhora significativa no número de iterações do método simplex. Para o desenvolvimento do método de expansão, considere o método primal simplex.

Considere uma "tolerância de trabalho" ε_t na iteração *t* do método do método simplex. Esta tolerância é ligeiramente incrementada antes de cada passo do método de simplex.

Escolhida a variável para entrar na base, digamos $\boldsymbol{x}_q$, e calculada as coordenadas básicas da direção primal simplex $\alpha_q = -\boldsymbol{B}^{-1}\boldsymbol{a}_{N_q}$ (sendo $\boldsymbol{B}$ a matriz base factível), o método de expansão inicia atualizando a tolerância de trabalho por: $\varepsilon_t = \varepsilon_{t-1} + \tau$ (com τ=4.9×10^{-11}, sugerido por Maros, 2003). Com esta nova tolerância calcula-se a seguinte razão:

$$\delta^* = min\left\{-\frac{x_{B_i} + \varepsilon_t}{\alpha_{iq}},\ i \in B,\ tal\ que\ \alpha_{iq} < -\varepsilon_z\right\}, \tag{3.1}$$

em que ε_z é a tolerância para verificar se a *i*-ésima componente da direção primal simplex é menor do que zero. Note que $\delta^* > 0$. Calcule também os valores que anulam as variáveis básicas,

$$\hat{\delta}_i = \left\{-\frac{\hat{x}_{B_i}}{\alpha_{iq}},\ i \in B,\ tal\ que\ \alpha_{iq} < -\varepsilon_z\right\}. \tag{3.2}$$

Defina S o conjunto dos índices das razões em (3.2) menores do que δ^*, isto é, $S = \left\{i \in B: \hat{\delta}_i \le \delta^*\right\}$.

Escolha um índice $p \in S$ (que define o índice da variável que sai da base) tal que $\left|\alpha_{pq}\right| \ge \left|\alpha_{iq}\right|\ \forall i \in S$. Esta escolha opta pelo maior pivô (α_{pq}) quando há empate na escolha da variável que sai da base (isto tende a melhorar a estabilidade numérica do método simplex).

Seja $\theta = \max\left\{\hat{\delta}_p, 0\right\}$. Observe que $\hat{\delta}_p$ é o passo necessário para zerar x_{B_p}. Até este ponto, o procedimento é conhecido como o *Teste das Razões de Harris* (1972), desenvolvido para que o pivô seja melhor escolhido, quando houver possibilidade.

O procedimento de *expansão* assegura que o tamanho do passo (menor passo aceitável) seja sempre diferente de zero e a função objetivo decresce, evitando a estagnação (mesmo que

teoricamente). Para isso, faça $\theta_{\min} = \frac{\tau}{\alpha_{pq}} > 0$ e o tamanho do passo θ_E é determinado por: $\theta_E = \max\{\theta, \theta_{\min}\}$.

A seguir, o resumo do procedimento descrito anteriormente.

Algoritmo - Método de Expansão (início da iteração t)
Passo 0: *INCREMENTO DA TOLERÂNCIA*

Faça $\varepsilon_t = \varepsilon_{t-1} + \tau$;

Passo 1: *PASSO EXPANDIDO*

Calcule $\delta^* = \min\left\{-\frac{x_{B_i} + \varepsilon_t}{\alpha_{iq}}, i \in B, \text{ tal que } \alpha_{iq} < -\varepsilon_z\right\}$;

Passo 2: *PASSO PIVÔ*

$p \leftarrow 0$; $\alpha_{\max} \leftarrow 0$;
Para *i* =1 ate *m* faça
Se $\alpha_{iq} < -\varepsilon_z$ então
$\hat{\delta}_i = -\frac{x_{B_i}}{\alpha_{iq}}$;
Se $\hat{\delta}_i \le \delta^*$ e $|\alpha_{iq}| > \alpha_{\max}$ então
$p = i$;
$\theta = \hat{\delta}_i$;
$\alpha_{\max} = |\alpha_{iq}|$;
fim do Se
fim do se
fim do para

Passo 3: *PASSO NÃO NULO*

Faça $\theta_{\min} = \frac{\tau}{\alpha_{pq}}$ e determine $\theta_E = \max\{\theta, \theta_{\min}\}$ (*tamanho do passo*)

Observe que se o passo usual para o maior elemento pivô é suficientemente maior do que zero, este é escolhido, senão um pequeno passo positivo é forçado.

Como em qualquer iteração do método, a variável não básica é aumentada por uma quantidade positiva na direção que melhora a função objetivo logo o valor da função objetivo nunca retorna a um valor anterior.

No método dual simplex com busca unidimensional (DSBU) as razões anulam uma variável básica dual e podem ser ultrapassadas, promovendo uma troca no sinal da variável básica e conseqüentemente mudança na taxa de variação na função objetivo dual (veja Sousa, 2005). Implementamos um procedimento simples, para o qual obtivemos os melhores resultados.

No desenvolvimento original do DSBU, caso ocorra violação primal no limitante superior, as razões $\frac{\lambda_{B_i}}{\eta_{iq}}$ (veja Sousa, 2005) são calculadas se λ_{B_i} e η_{iq} têm sinais opostos. Para o caso de ocorrer a violação primal no limitante inferior, as razões são calculadas se λ_{B_i} e η_{iq} têm mesmo sinal.

A única mudança que realizamos no desenvolvimento original de DSBU foi calcular essas razões utilizando-se a tolerância de trabalho ε_t (incrementado por $1e$-7) a cada iteração do método e a busca unidimensional faz a escolha de qual variável deixa base (Sousa, 2005).

Tentamos outras variações neste procedimento para que tivéssemos um método de expansão como o original, como por exemplo, ao realizar a busca unidimensional, fosse levado em conta o tamanho do elemento pivô, mas não obtivemos bons resultados. Na próxima seção relatamos os experimentos computacionais.

4. Experimentos Computacionais

4.1. Introdução

Apresentamos agora, os resultados obtidos usando como problemas testes os quais foram relatados por Gass e Vinjamuri (2004) citado na seção 2.

Na Tabela 4.1 são apresentados os resultados computacionais obtidos com o CPLEX e método dual simplex com busca unidimensional

Tabela 4.1 – Experimentos computacionais de problemas com ciclagem

		iterações		
problema	***nome***	***CPLEX***[2]	***DSBU***[3]	***FO***[4]
1	***Hoffman***	*2*	*2*	*0.00*
2	***Beale***[1]	*3*	*3*	*-0.05*
3	***Yudin***	*3*	*3*	*0.50*
4	***Yudin2***	*6*	*4*	*-1.00*
5	***Kuhn***	*4*	*5*	*-2.00*
6	***Marshall***	*0*	*0*	*0.00*
7	***Marshall2***	*4*	*4*	*-2.00*
8	***Solow***	*3*	*2*	*0.00*
9	***Sierksma***	*2*	*2*	*ilimitado*
10	***Chvátal***	*4*	*3*	*1.00*
11	***Nering***[1]	*4*	*3*	*ilimitado*

[1] Problemas com os dados fracionários;
[2] CPLEX 7.5 com o método dual simplex;
[3] DSBU: Método dual simplex com busca unidimensional;
[4] FO: Valor da função objetivo.

Observamos que, apesar do método dual simplex com busca unidimensional, neste caso, não se utilizar de nenhum procedimento anti-degeneração, a solução de cada problema foi encontrada em número menor de iterações do que o CPLEX (exceto exemplar 5). Os autores também já haviam relatado que os principais pacotes de otimização resolvem todos os exemplos construídos sem ciclagem, o que de fato foi observado para alguns problemas apresentados por Zornig (2006).

Daremos a seguir os resultados computacionais obtidos com o método DSBU utilizando o procedimento de expansão adaptado para tratar degeneração em alguns problemas da Netlib e gerados aleatoriamente.

4.2. Método DSBU com o Método Expansão Adpatado

Apresentamos a seguir os resultados para o método dual simplex com busca unidimensional para resolver alguns problemas degenerados da Netlib e outros gerados aleatoriamente na forma canalizada. Os testes foram realizados em um computador de 2.80 GHz com 2.0 GBytes de RAM com o tempo de processamento medido em segundos.

Os problemas da Netlib que trabalhamos são problemas pequenos mas são problemas difíceis de serem resolvidos, no sentido que são muitos sensíveis a qualquer perturbação.

A Tabela 4.2 apresenta os resultados obtidos para os problemas pequenos da Netlib para o método DSBU usual (sem nenhum procedimento anti-degeneração) e com os aplicativos computacionais CPLEX 7.5 e XPRESS , utilizando-se a regra de Dantzig (ou regra usual). O símbolo % representa a porcentagem de elementos não-nulos na matriz de restrições que definida por *m*'restrições e *n* variáveis. Para o pacote CPLEX 7.5, o pré-processador não foi utilizado, para que a comparação seja 'mais justa'. Vale observar que o pré-processamento é uma ferramenta poderosa para eliminar redundâncias (por conseguinte, degeneração), fixar variáveis, apertar limitantes de modo que o problema resultante é mais fácil de ser resolvido, que será explorado em trabalhos futuros.

Tabela 4.2 – Problemas da Netlib

problemas			***DSBU (usual)***		***XPRESS***		***CPLEX***	
nome	***%***	***m'×n***	***iteração***	***tempo***	***iteração***	***tempo***	***iteração***	***tempo***
afiro	*8.30*	***25×39***	*18*	*0.00*	*15*	*0.01*	*11*	*0.00*
adlitlle	*2.49*	***53×196***	*109*	*0.10*	*32*	*0.02*	*125*	*0.00*
kb2	*12.79*	***43×52***	*227*	*0.01*	*20*	*0.01*	*28*	*0.00*
sc50a	*5.52*	***49×48***	*52*	*0.00*	*59*	*0.01*	*45*	*0.00*
sc50b	*5.12*	***48×48***	*50*	*0.00*	*56*	*0.02*	*53*	*0.00*

Podemos observar que o método DSBU (usual) teve o pior desempenho, sendo que o pacote CPLEX foi na média ligeiramente o melhor, no entanto o aplicativo XPRESS realizou apenas 32 iterações para o problema *adlittle*, um ganho muito significante.

Na Tabela 4.3 reportamos o número de iterações para o método DSBU usual, DSBU com o método expansão adaptado para tratar degeneração (Sousa, 2005) e para o CPLEX.

Tabela 4.3 – Problemas da Netlib (número de iterações)

	DSBU	***DSBU***	***CPLEX***
nome	***usual***	***expansão***	
afiro	*18*	*15*	*11*
adlitlle	*109*	*108*	*125*
kb2	*227*	*31*	*28*
sc50a	*52*	*51*	*45*
sc50b	*50*	*50*	*53*

A Tabela 4.3 mostra que o método de expansão fez reduzir o número de iterações nos problemas afiro, adliitle, sc50a e principalmente para o problema *kb2*, se aproximando do CPLEX que deve ter técnicas sofisticadas para tratar degeneração.

Ressalta-se que o tempo para DSBU com o método de expansão, foi praticamente o mesmo que DSBU usual, já que não utiliza nenhum procedimento computacionalmente caro.

A seguir, apresentamos os resultados com o método DSBU em problemas gerados aleatoriamente, para verificar seu desempenho com o procedimento de expansão, mas antes, exibimos a estrutura da matriz de restrições.

Para cada tamanho do problema, que define um exemplar com *m'* restrições e *n* variáveis (veja Figura 1), os dados foram gerados aleatoriamente.

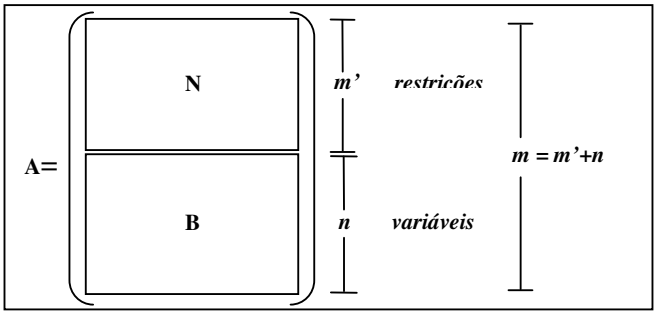

Figura 1 - Representação da matriz na forma geral (problema 1.1)

Os resultados obtidos foram para uma classe esparsa de problemas, para os quais a matriz de restrição é da forma escada, veja Figura 2.

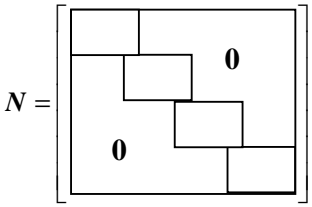

Figura 2 - Representação da matriz ***N*** não-básica na forma escada

Os coeficientes não-nulos da matriz **A** estão confinados nos blocos e foram gerados aleatoriamente. Considere nas tabelas 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 (Sousa, 2005). O número de iterações, reportados a seguir, são as médias dos resultados da resolução dos 20 exemplares para cada tamanho (dimensão) do problema.

A Tabelas 4.4 apresenta os resultados com DSBU usual e DSBU expansão, respectivamente, para os problemas degenerados, que foram gerados aleatoriamente, para que possamos comparar melhor se o método de expansão diminui a estagnação no número de iterações.

Tabela 4.4 – Problemas degenerados resolvidos pelo DSBU-usual e DSBU-expansão

problemas						***iterações***		***tempo***	
nb	***m1***	***n1***	***cc***	***%***	***m'×n***	***usual***	***expansão***	***usual***	***expansão***
20	*1*	*20*	*0*	*5.00*	***20×400***	*19.8*	*19.8*	*0.11*	*0.11*
100	*2*	*5*	*3*	*2.46*	***200×203***	*199.6*	*189.8*	*0.78*	*0.71*
25	*8*	*16*	*4*	*5.26*	***200×304***	*36.3*	*31.8*	*0.40*	*0.36*
50	*4*	*12*	*3*	*2.64*	***200×453***	*12.8*	*12.4*	*0.33*	*0.32*
200	*2*	*6*	*4*	*1.48*	***400×404***	*532.2*	*486.0*	*24.6*	*18.63*

Os resultados mostram que o método de expansão faz reduzir o número de iterações para estes problemas degenerados, como conseqüência, obviamente o tempo de resolução foi menor, já que o procedimento de expansão na acarreta em custo computacional.

Esta redução fica mais evidente nos problemas (400×404), neste caso o mais esparso, em que o número de iterações foi 10% e 15% menor (em relação DSBU usual).

Diante destes resultados, utilizamos o pacote CPLEX para resolver esta mesma classe de problemas, cujos resultados estão na Tabela 4.5.

Tabela 4.5 – Problemas degenerados resolvidos pelo CPLEX

problemas						***CPLEX***	
nb	***m1***	***n1***	***cc***	***%***	***m'×n***	***iteração***	***tempo***
20	*1*	*20*	*0*	*5.00*	***20×400***	*73.5*	*0.00*
100	*2*	*5*	*3*	*2.46*	***200×203***	*257.2*	*0.00*
25	*8*	*16*	*4*	*5.26*	***200×304***	*50.1*	*0.00*
50	*4*	*12*	*3*	*2.64*	***200×453***	*17.8*	*0.00*
200	*2*	*6*	*4*	*1.48*	***400×404***	*603.4*	*0.04*

Comparando os resultados das Tabelas 4.4 e 4.5, fica clara a superioridade do método DSBU frente ao CPLEX, principalmente quando se faz o uso do método de expansão, embora o tempo não seja favorável. Por exemplo, para os problemas 20×400, o pacote CPLEX realizou aproximadamente 3 vezes mais iterações do que o método DSBU, logo é importante observar que nestes problemas tendo apenas 20 restrições, 80% dos custos são iguais a zero, caracterizando-se, assim, problemas muitos degenerados como os problemas 400×404, mas que para estes problemas não há uma diferença (de 3 vezes mais iterações) por se tratar de exemplares bem mais restrito, ou seja, a degeneração é um problema que deve ser tratado com métodos eficientes.

Segundo informações do manual do CPLEX , problemas degenerados são tratados com perturbação, mas notamos que o método de expansão desenvolvido para o DSBU apresentou resultados melhores para os problemas degenerados que foram gerados aleatoriamente.

Portanto, fica evidente a necessidade de se utilizar algum procedimento anti-degeneração no método simplex para resolver problemas degenerados.

5. Conclusões e Trabalhos Futuros

Os experimentos computacionais revelam uma pequena vantagem do método DSBU frente ao CPLEX do ponto de vista de iterações em problemas degenerados, mas inferior no tempo computacional.

A degeneração e ciclagem no método simplex estão intimamente relacionados, mas quando se resolve computacionalmente problemas com estas características, a ciclagem não é um problema relevante. Por outro lado, observamos que pelos primeiros testes computacionais que a degeneração é um fenômeno que provoca a estagnação do método simplex. Daí, a necessidade de resolver mais problemas da Netlib e outros de maior tamanho gerados aleatoriamente, que serão alvos de trabalhos futuros.

6. Referências Bibliográficas

Avis, D., Chvátal, V. (1978) "Notes on Bland's Pivoting Rule" *Mathematical. Programming Study* 8, 24-34.

Beale, E. (1955) "Cycling in the Dual Simplex Method" *Naval Research Logistics Quarterly*, 2, 4, 269-275.

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.

Bland, R. G. (1977) "New Finite Pivoting Rules For the Simplex Method" *Mathematical Operations Research*., 2, 2, 103-107.

Charnes A. (1952) Optimality and degeneracy in linear programming. *Econometrica*, 20:160–70.

Dantizg, G., Orden, A., Wolfe, P. (1955) "A Generalized Simplex Method for Minimizing a Linear Form under Linear Inequality Constraints" *Pacific Journal of Mathematics*, 5, 2, 183-195.

Elhallaoui, I., Metrane, A., Desaulniers, G., Soumis, F. (2008) An improved primal simplex algorithm for degenerate linear programs. *Operations Research*, submitted for publication.

Gass, S. I. (1993) "Encounters with Degeneracy: a Personal View" *Annals of Operations Research*, 47, 335 –42.

Gass, S. I. and Vinjamuri, S. (2004) "Cycling in linear Programming Problems" *Computers and Operations Research*, 31, 303-311.
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.
Harris, P.M. (1973) "Pivot Selection Method of the Devex LP Code" *Mathematical Programming* 5, 1-28.
Hoffman, A. J. (1953) "Cycling in the Simplex Algorithm" Washington, DC, *National Bureau of Standards*.
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.
Pan P-Q. (1998) A basis deficiency-allowing variation of the simplex method for linear programming. *Computers and Mathematics with Applications,* 36(3): 33–53.
Ryan, D. and Osborne, M. (1988) "On the Solution of Highly Degenerate Linear Programmes" *Mathematical Programming*, 41, 385-392.
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. (2000) "*Estudos em Otimização Linear*" Dissertação de Mestrado, ICMC-USP.
Sousa, R. S. (2005) *Métodos Tipo Dual Simplex para Problemas de Otimização Linear Canalizados*, Tese de Doutorado, ICMC-USP.
Sousa, R. S., Silva, C. T. e Arenales, M. N. (2005) Métodos do Tipo Dual Simplex para Problemas de Otimização Linear Canalizados. *Pesquisa Operacional*, v 25, n.3, 349-382.
Sousa, R. S., Oishi, C. M. e Arenales, M. N. (2006) Método Dual Simplex com Busca Unidimensional Utilizando o Gradiente Bi-Conjugado. Anais do XXXVIII SBPO *Simpósio Brasileiro de Pesquisa Operacional*.
Raymond, V., Soumis, F., Orban, D. (2010) A new version of the Improved Primal Simplex for degenerate linear programs. *Computers & Operations Research* 37, 91-98.
Terlaky, T. (2000) "*Linear Programming, Simplex-Type Algorithms*" Department of Computing and Software, McMaster University, Hamilton, ON, Canada. http://www.cas.mcmaster.ca/~terlaky
Vanderbei, R. J., *Linear Programming: Foundations and Extensions*, Kluwer Academic Publishers, 1997.
Yudin, D. B., Golshtein, E. G. (1965) "*Linear Programming*" Israel Program of Scientific Translations, Jerusalem.
Zörnig P. (2006) Systematic construction of examples for cycling in the simplex method. *Computers & Operations Research*. 33(8):2247–62.
Zörnig P. (2008) A note on cycling LP examples with permutation structure. *Computers & Operations Research*. 35; 994 – 1002.