Working Paper

A penalty based simplex method for linear programming

Artur Świętanowski

WP-95-005
January 1995

A penalty based simplex method for linear programming

Artur Świętanowski

WP-95-005
January 1995

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 807 □ Fax: +43 2236 71313 □ E-Mail: info@iiasa.ac.at

Abstract

We give a general description of a new advanced implementation of the simplex method for linear programming. The method "decouples" a notion of the simplex *basic solution* into two independent entities: *a solution* and *a basis*. This generalization makes it possible to incorporate new strategies into the algorithm since the iterates no longer need to be the vertices of the simplex. An advantage of such approach is a possibility of taking steps along directions that are not simplex edges (in principle they can even cross the interior of the feasible set). It is exploited in our new approach to finding the initial solution in which global infeasibility is handled through a dynamically adjusted penalty term.

We present several new techniques that have been incorporated into the method. These features include:

- previously mentioned method for finding an initial solution,
- an original approximate steepest edge pricing algorithm,
- dynamic adjustment of the penalty term.

The presence of the new crashing and restart procedures based on the penalty term make the algorithm particularly suitable for sequential "warm start" calls when solving subproblems in decomposition approaches. The same features may be used in post optimal analysis.

The efficiency of the new features is demonstrated when running the method on a subset of difficult linear programs from the NETLIB collection of Gay [7].

Key words: simplex method, linear penalty, crashing, steepest edge pricing.

Contents

A penalty based simplex method for linear programming[1]

Artur Świętanowski[2]

1. Introduction

We are concerned with solution of a linear optimization problem of the form

$$\min \mathbf{c}^T\mathbf{x} \tag{1.1}$$

subject to

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{l} \le \mathbf{x} &\le \mathbf{u} \end{aligned} \tag{1.2}$$

where $\mathbf{A} \in \Re^{m\times n}$, $\mathbf{b} \in \Re^m$, $\mathbf{x}$, $\mathbf{c}$, $\mathbf{l}$, $\mathbf{u} \in \Re^n$, (some l_j or u_j may be inifinite).

There exists a number of comercially available high quality simplex type linear optimizers. The revised simplex method by Dantzig ([4]) was being developed ever since its introduction in 1963 by numerous pure researchers as well as practicioners. Nevertheless, we felt compelled to produce yet another implementation of this well-known method. There were numerous reasons, of which we choose to name here a need for a restarting primal simplex algorithm to be later used in a decomposition scheme (see [16]) and some new parallel approaches to linear programming (see [20]).

We shall present a modified version of a primal simplex algorithm which we think, apart from its general applicability, is perfectly suitable for decomposition schemes requiring efficient restarts of the algorithm. We call our simplex code "penalty based" because the use of penalty in objective function instead of a two-phase method is the key to method's ability to start (and restart) from any point. The new and original techniques incorporated in our code and described in this paper include:

1. replacing of the notion of the *simplex basic solution* with two separate entities: *a solution* and *a basis*:

   By dropping the requirement that each simplex method iterate has to be a so-called basic solution (one in which all non-basic variables are kept on their bounds) we have much more flexibility when choosing the initial solution (which we use to our advantage in the crashing procedure) and, possibly, when forming all other iterates.

2. a consistent approach to using penalty function:

   The penalty is introduced in order to deal with infeasibility of the initial solution (either the one produced by the crashing procedure, or one provided by the user). Throughout all iterations of the method it is kept under control and dynamically adjusted when necessary. A possible infeasibility of the linear problem is detected and proven.

[1]This research was partially sponsored by the Committe for Scientific Research of Poland grant no. PB 8 S505 015 05. Parts of it were done during author's stay in the International Institute for Applied Systems Analysis in Laxenburg, Austria.

[2]Institute of Automatic Control & Computation Engineering, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland.

3. a new and stunningly efficient approximation of steepest edge pricing:

   We show that what is possibly the simplest steepest edge approximation is also a very efficient one.

This paper is also intended to document our implementation of the revised simplex method. Obviously, we will focus our attention on the new ideas tested in our code, but a limited description of features which were previously known will also be presented. Since the efficiency of any modern linear optimizer is of paramount importance we outline the main algorithmic techniques that make our implementation of the simplex method one of the most advanced currently available.

In Section 2. we shortly present the textbook form of the revised simplex method algorithm. The algorithmic and programming techniques which make the simplex method one of the most efficient linear programming methods on the market are outlined in Section 3.. Although two of the three pricing schemes we employ are now "classical", we decided to describe them in a separate Section 4. in which we derive our own pricing algorithm. Section 5. presents our rather lax approach to the notions of basic and non-basic variables and solutions as well as explains the dynamic penalty method rationale and implementation. Our claim that the penalty-based simplex can easily be restarted is proven in Section 6.. Finally, in Sections 7. and 8. the analysis of numerical experiments conducted on the subset of Gay's NETLIB test problem collection (see [7]) and our conclusions regarding the practicability and usefulness of the techniques proposed are given.

2. The basic algorithm of the revised simplex method

Let us assume that initial partition of the constraint matrix $\mathbf{A}$ into a non-singular basis matrix $\mathbf{B}$, $\mathbf{B} \in \Re^{m \times m}$ and the non-basic part $\mathbf{N}$, $\mathbf{N} \in \Re^{m \times (n-m)}$ is known. We then have

$$\mathbf{A} = [\mathbf{B}\ \ \mathbf{N}],\ \mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix},$$
$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_B \\ \mathbf{c}_N \end{bmatrix},\ \mathbf{l} = \begin{bmatrix} \mathbf{l}_B \\ \mathbf{l}_N \end{bmatrix},\ \mathbf{u} = \begin{bmatrix} \mathbf{u}_B \\ \mathbf{u}_N \end{bmatrix}$$

Additionally, let us assume that an initial feasible solution

$$\begin{gathered} \mathbf{l}_N \le \mathbf{x}_N^0 \le \mathbf{u}_N, \\ \mathbf{x}_B^0 = \mathbf{B}^{-1}(\mathbf{b} - \mathbf{N}\mathbf{x}_N), \\ \mathbf{l}_B \le \mathbf{x}_B^0 \le \mathbf{u}_B \end{gathered}$$

is known as well. We shall now proceed to recall a basic version of the revised simplex method. (For an in-depth discussion of the simplex method see e.g. Dantzig [4], Nazareth [11], Murtagh [10], Forrest and Tomlin [6] and many others.) Note that steps 3 and 5 are formulated so as to allow the non-basic variables to be *between* their bounds and not on them. This is necessary when some of the variables (called free) have two infinite bounds. Later it will be shown to have other uses as well.

Algorithm I: The basic algorithm of the revised simplex method

1. Compute the dual variables:

$$\pi = \mathbf{B}^{-T}\mathbf{c}_B$$

2. Compute the vector of reduced costs $\mathbf{z}$:

$$z_j = \begin{cases} 0 & j = 1, \dots, m \\ c_j - \pi^T \mathbf{a}_j & j = m+1, \dots, n \end{cases}$$

3. Choose the most favourable reduced cost z_q and test optimality:

$$q = \arg \min_{i=m+1,\dots,n} \begin{cases} z_i & x_i = l_i, \\ -z_i & x_i = u_i, \\ -|z_i| & l_i < x_i < u_i \end{cases}$$

   If $z_q \geq 0$ then the current solution is optimal.

4. Compute the basic variables' change direction η:

$$\eta_q = \mathbf{B}^{-1}\mathbf{a}_q$$

5. Calculate a feasible steplength θ and check for unboundedness.

   If θ is smaller than the distance between the current value of the q-th variable and the finite bound towards which it is moving then the q-th variable is shifted towards appropriate bound, the basis is not changed and a cheap simplex iteration is performed.

   If $\theta = +\infty$ then declare the linear problem (1.1)–(1.2) unbounded.

6. Make the step and revise the basis:

$$\mathbf{x}_B^{i+1} = \mathbf{x}_B^i - \theta\eta_q$$

   Exchange columns p and q of the constraint matrix $\mathbf{A}$ and rows p and q of column vectors $\mathbf{x}$, $\mathbf{c}$, $\mathbf{l}$, $\mathbf{u}$. After such permutation the first m columns of $\mathbf{A}$ will again constitute a non-singular basic matrix.

   Go back to step 1.

3. Algorithmic enhancements

The revised simplex algorithm described above creates a framework for actual implementation of the method. After years of research and development of this algorithm virtually every step is performed differently than straightforward mathematical formulas would suggest. Detailed description of origin, meaning and computational gains of all the advanced techniques incorporated in our code is far beyond the scope of this paper. Interested reader is referred to an excellent (and very much implementation oriented) book of Nazareth [11] for a comprehensive analysis of the workings of the simplex method. Our own experience with implementing simplex was summarized in Świętanowski [17] and [18]. Other references will be given as we mention particular techniques used.

3.1 Taking advantage of problem sparsity

In order to be able to solve practical problems with thousands (or tens of thousands) of constraints and variables, we exploit sparsity of the constraint matrix $\mathbf{A}$ by storing only the non-zero entries of $\mathbf{A}$ and an LU factorization of the current simplex basis $\mathbf{B}$. Our factorization is supplemented with Bartels-Golub update procedure [1] with enhancements described in Reid [12]. For an excellent overview of an implementation of both LU factorization and updates see also Suhl and Suhl [15].

Of considerable significance is the fact that in modern computers memory is plentiful and inexpensive, especially in comparison with processing power. Therefore instead of carefully balancing storage requirements and amount of computations we now are more inclined to use up much more memory to gain (sometimes moderate) savings in computation time. One of the ideas taking advantage of this change in computer hardware is duplicate storage of the constraint matrix. It is now stored not only by columns (as a file of packed columns), which was suitable for column oriented algorithms of the past. We also have an independent row-wise representation of it. This idea is quite new indeed. It has been incorporated in our simplex code in September 1993. The same idea is put forward by Robert Bixby in his introductory article in ORSA Journal on Computing [3]. It is used during crashing (initial basis construction) and, more importantly, when updating reduced costs and steepest edge weights (exact or approximate).

The basic idea behind the duplicate constraint matrix representation is that reduced costs, dual variables (as in Bixby's CPLEX [3]) or steepest edge weights may be updated much more efficiently when using the row-wise representation. During the early phases of developing our implementation (when the row-wise representation was being introduced) we have seen a reduction of problem solution time by approximately 20% to 40%.

3.2 Problem scaling

In lack of a commonly agreed upon scaling quality criterion we have decided to use a rather "fuzzy" definition. Matrix **A** is considered poorly scaled if its non-zero entries are of vastly different magnitudes.

In order to avoid numerical difficulties caused by poor scaling of the constraint matrix we use a simple two pass scaling scheme followed by column equilibration. By column equilibration we mean dividing each column of the constraint matrix by the norm of its largest non-zero. After this operation the largest non-zero in the matrix does not exceed unity. For an overview of some linear problem scaling techniques and assessment of their impact on the simplex method see Tomlin [19].

The scaling technique is modified in order to assure that scaling itself does not introduce any roundoff error. This is achieved by using only integer powers of two as scaling factors. We compute "ideal" scaling factors and then use their approximation by powers of two (see Świętanowski [17]). This approach has one additional advantage: reduction of computational effort needed to scale the problem (and later retrieve solution for the original LP from the solution to a scaled one). Instead of multiplying or dividing numbers by scaling factors we only shift the binary mantissa.

3.3 Initial basis

The crashing method implemented in our program is based on an idea of Bixby [2]. We divide the variables into so-called preference sets, and then build the basis using as many columns of the original constraint matrix as possible. Our algorithm is simplified by requirement that a (permuted) triangular basis always has to be found. When the process is finished without producing a complete basis, the missing places are filled with unity columns and their corresponding artificial variables are added.

3.4 Pivoting strategy

Last but not least, we shortly describe the rules we use for selecting the variable which leaves the basis. Again, the standard method known from the basic version of the simplex algorithm is modified. As it was proposed by Harris [9], we first calculate the *maximum steplength* in the perturbed problem (one with slightly expanded simple bounds on basic variables). We then choose the largest (and thus the most stable numerically) pivot which will not exceed the steplength in original ("exact") bounds. This technique has been shown to decrease the likelihood of arriving at singular bases, which is a phenomenon commonly known to appear during solution of numerically difficult problems.

4. Efficient pricing techniques

Many authors have pointed out that the method for selection of the variable which enters the basis is of crucial importance for the simplex method's performance. It has long been known that the "most negative reduced cost" criterion of Dantzig is not always efficient. Our code allows not only Dantzig's pricing method, but also a steepest edge algorithm of Goldfarb and Reid [8] as well as our own approximation of it. For a survey of a number of steepest edge algorithms consult Forrest and Goldfarb [5].

4.1 Steepest edge rationale

The standard pricing method chooses the variable which guarantees the largest objective change per unit move along the axis. Steepest edge (SE) approach prefers the variable wich yields the largest objective decrease per unit move along the actual edge of the simplex. Instead of comparing reduced costs

$$z_j = c_j - \mathbf{c}_B^T \eta_j$$

we compare normalized reduced costs

$$\tilde{z}_j = \frac{c_j - \mathbf{c}_B^T \eta_j}{\sqrt{\|\eta_j\|^2 + 1}}$$

where $\|\cdot\|$ denotes Euclidean norm.[3] Explicit computation of all (or even some) of the norms $\|\eta_j\|$ in each simplex iteration would be prohibitively expensive. We can however derive reccurrences for updating the sqares of norms of direction vectors.

We now proceed to recall the basic recurrences used by Goldfarb and Reid [8] to update the steepest edge *weights* $\gamma_j = \|\eta_j\|^2 + 1$. This will allow us to derive and explain the recurrences used in our new approximate steepest edge algorithm.

Let $\mathbf{T}$ represent the expanded simplex tableau

$$\mathbf{T} = \begin{bmatrix} \mathbf{B}^{-1} & -\mathbf{B}^{-1}\mathbf{N} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{4.1}$$

$$\mathbf{t}_j = \mathbf{T}\mathbf{e}_j, \; j \in \{m+1, \ldots, n\}$$

and α — the p-th (pivotal) row of $\mathbf{T}$

$$\alpha = \mathbf{T}^T \mathbf{e}_p = \begin{bmatrix} \mathbf{B}^{-T}\mathbf{e}_p \\ -\mathbf{N}^T\mathbf{B}^{-T}\mathbf{e}_p \end{bmatrix}, \; p \in \{1, \ldots, m\}.^4 \tag{4.2}$$

Note that vectors $\mathbf{t}_j$ may be expressed using direction vectors η_j of the basic simplex algorithm as

$$\mathbf{t}_j = \begin{bmatrix} \eta_j \\ \mathbf{e}_{j-m} \end{bmatrix}, \; j = m+1, \ldots, n$$

The quantities with a tilde will refer to values *after* the exchange of basic column p and non-basic column q. The direction vectors $\mathbf{t}_j$ are updated according to the following formulas:

$$\begin{aligned} \tilde{\mathbf{t}}_q &= \mathbf{t}_q \frac{1}{\alpha_q} \\ &\text{and for } j = m+1, \ldots, n, \; j \neq q \\ \tilde{\mathbf{t}}_j &= \mathbf{t}_j - \mathbf{t}_q \frac{\alpha_j}{\alpha_q} = \\ &\quad \mathbf{t}_j - \mathbf{t}_q \tilde{\alpha}_j. \end{aligned} \tag{4.3}$$

[3] Note the alternative equation used for computing the reduced cost.

[4] Please, note that $\mathbf{e}_p$ always denotes a vector of appropriate dimension. When used in expression $\mathbf{T}\mathbf{e}_p$ it is $\mathbf{e}_p \in \Re^n$, later in e.g. $\mathbf{B}^{-T}\mathbf{e}_p$ it is in $\Re^m$.

The weights

$$\gamma_j = \|\mathbf{t}_j\|^2 = \mathbf{t}_j^T\mathbf{t}_j = \eta_j^T\eta_j + 1,\ j = m+1,\dots,n$$

are updated as follows:

$$\begin{aligned}\tilde{\gamma}_j &= (\mathbf{t}_j - \tilde{\alpha}_j\mathbf{t}_q)^T(\mathbf{t}_j - \tilde{\alpha}_j\mathbf{t}_q)\\ &= \mathbf{t}_j^T\mathbf{t}_j - 2\tilde{\alpha}_j\mathbf{t}_j^T\mathbf{t}_q + \tilde{\alpha}_j^2\mathbf{t}_q^T\mathbf{t}_q\\ &= \gamma_j - 2\tilde{\alpha}_j\mathbf{t}_j^T\mathbf{t}_q + \tilde{\alpha}_j^2\gamma_q\\ &= \gamma_j - 2\tilde{\alpha}_j\eta_j^T\eta_q + \tilde{\alpha}_j^2\gamma_q.^5\end{aligned} \tag{4.4}$$

The reduced costs may be updated, which is much cheaper than computation of dual variables and reduced costs proposed in the basic algorithm of Section 2.. The update is performed according to a formula known from the tableau form of the simplex algorithm:

$$\begin{gathered}\tilde{z}_q = z_q/\alpha_q\\ \tilde{z}_j = z_j - \tilde{\alpha}_q z_q,\ j = m+1,\dots,n,\ j \neq q.\end{gathered} \tag{4.5}$$

4.2 Steepest edge simplex algorithm

We now present a simplex algorithm in which the reduced costs are updated in each iteration (instead of being computed afresh) and optionally steepest edge pricing may be performed.

Algorithm II: Simplex algorithm with steepest edge weights and reduced cost updates

0. Initialization of the algorithm:

   Compute dual variables

   $$\pi = \mathbf{B}^{-T}\mathbf{c}_B$$

   and a vector of reduced costs $\mathbf{z}$

   $$z_j = \begin{cases} 0 & j = 1,\dots,m\\ c_j - \pi^T\mathbf{a}_j & j = m+1,\dots,n.\end{cases}$$

   Reset steepest edge weights γ. Since it is impractical (too expensive) to compute for all non-basic variables the exact norms of their corresponding direction vectors, we have decided to assume that the linear problem constraint matrix columns are scaled and equilibrated according to our default scaling scheme. Thus, every time we reset the steepest edge weights, we set γ_j to be equal to the number of non-zero entries of the j-th column of the constraint matrix plus one. This promotes shortest (sparsest) columns and encourages construction of sparse bases.

1. Choose the most favourable weighted reduced cost z_q and check for optimality:

   Identical to the standard simplex algorithm except that when steepest edge is employed reduced costs z_j are weighted (divided by square roots of weights γ_j).

   The in-coming column number q is found or optimality of the current solution is detected.

2. Compute change direction η of the basic variables:

   $$\eta = \mathbf{B}^{-1}\mathbf{a}_q$$

3. Compute z_q and γ_q afresh:

   $$\begin{gathered}z_q = c_q - \mathbf{c}_B^T\eta\\ \gamma_q = \eta^T\eta + 1\end{gathered}$$

[5] For any $i \neq j$ $\mathbf{t}_i^T\mathbf{t}_j = \eta_i^T\eta_j$.

4. Confirm that move in direction η_q is profitable:

   Since the vector of reduced costs $\mathbf{z}$ is updated, it accumulates roundoff errors. Therefore we need to verify the sign of z_q. If the verification fails we go to step 0.

5. Calculate steplength θ and check for unboundedness:

   Identical as in the basic algorithm. Stop if problem unbounded. Determine pivot row number p and *store separately* the pivot element η_p (which is equal to α_q).

6. Make the step:

$$\tilde{\mathbf{x}}_B = \mathbf{x}_B - \theta\eta$$

7. Calculate work vector β (needed only for steepest edge pricing):

$$\kappa = \mathbf{B}^{-T}\eta$$
$$\beta = \mathbf{A}^T\kappa$$

8. Update the basis representation:

$$\tilde{\mathbf{B}} = \mathbf{B} + \mathbf{e}_p(\mathbf{a}_q - \mathbf{a}_p)^T$$

9. Compute the value of the pivot row p of the simplex tableau for the *next* iteration:

$$\delta = \tilde{\mathbf{B}}^{-T}\mathbf{e}_p$$

$$\tilde{\alpha} = \begin{bmatrix} \mathbf{B}^{-T} \\ -\mathbf{N}^T\mathbf{B}^{-T} \end{bmatrix} \mathbf{e}_p = \begin{bmatrix} \mathbf{I} \\ -\mathbf{N}^T \end{bmatrix} \delta$$

10. Update the reduced costs:

$$\tilde{z}_q = z_q/\alpha_q$$
$$\tilde{z}_j = z_j - \tilde{\alpha}_j z_q,\ j = m+1,\ldots,n \text{ and } j \neq q$$

11. Update the steepest edge weights:

$$\begin{aligned} \tilde{\gamma}_q &= \gamma_q/\alpha_q^2 \\ \tilde{\gamma}_j &= \max(\gamma_j - 2\tilde{\alpha}_j\beta_j + \tilde{\alpha}_j^2\gamma_q,\ \tilde{\alpha}_j^2 + 1),\ j = m+1,\ldots,n \text{ and } j \neq q \end{aligned} \tag{4.6}$$

12. Go back to step 1.

Note that steps 6 and 8 may be performed more efficiently if row-wise representation of the constraint matrix is available. Vector κ may be quite sparse and vector δ (which is a row of the basis matrix inverse) is almost certainly *very* sparse. Our experience indicates that regardless of the problem's dimension, it usually has only a few non-zeros. If we use the row representation we may scan only those rows of the constraint matrix $\mathbf{A}$ which correspond to non-zeros in vectors δ and κ respectively. The reduction in computation time is impressive (especially when only the reduced costs are updated and steepest edge pricing is not performed).

Emphasis should be given to the fact that steepest edge pricing is in principle more expensive than standard (Dantzig's) pricing, especially if the latter is either performed as partial or multiple pricing (see e.g. Nazareth [11]) or when reduced costs' or dual variables' updates are done. This setback is partially compensated by decrease in the avarage number of iterations needed to solve a linear problem when a superior pricing technique is used.

Let us now examine additional operations needed to compute and update steepest edge weights. The computation of the steepest edge weights needs the following arithmetic operations:

- A division of reduced costs by the weights during pricing.
- Computation of β. To this end one BTRAN and then a single pass through the matrix **A** are needed. The latter is fairly expensive since the intermediate result κ may be dense.
- Weights update according to formula 4.6. It requires neither a pass through the constraint matrix nor a linear system solution.

From the foregoing it is clear that computation of vector β (needed to calculate $\eta_j^T\eta_q$) is the single most expensive task performed.

4.3 Approximate steepest edge strategy

Recall the formula we use to calculate exact weights:

$$\begin{aligned}\eta_j^T\eta_q &= \mathbf{a}_j^T\mathbf{B}^{-T}\eta_q = \mathbf{a}_j^T\kappa,\\ \kappa &= \mathbf{B}^{-T}\eta_q.\end{aligned}$$

We can reduce the computational effort, avoid computing vector β and use approximate weights ϕ instead. Observe that vectors $\mathbf{t}_j$, $j = m+1,\ldots,n$ are the columns of the expanded tableau **T** while vector α is its row (compare with formula (4.1)). We already know one non-zero of each η_j. Thus the product

$$\eta_j^T\eta_q = \alpha_j\alpha_q + \sum_{i=1,\ldots,m i\neq p} (\eta_j)_i(\eta_q)_i$$

may be (very roughly) approximated by

$$\eta_j^T\eta_q = \alpha_j\alpha_q = \tilde{\alpha}_j\alpha_q^2$$

if we assume that the vectors η_j are quite sparse and thus most of the products $(\eta_j)_i(\eta_q)_i$ are zero. At the same time we know a lower bound on *exact* weight γ_j

$$\gamma_j \geq (\eta_j)_p^2 + 1 = \alpha_j^2 + 1 = \tilde{\alpha}_j^2\alpha_q^2 + 1.$$

Therefore we may update the approximate weights ϕ_j for $j \neq q$

$$\tilde{\phi}_j = \max(\phi_j, \tilde{\alpha}_j^2\alpha_q^2 + 1) - 2\tilde{\alpha}_j^2\alpha_q^2 + \tilde{\alpha}_j^2\gamma_q \tag{4.7}$$

and we may still use the exact formula for $j = q$

$$\tilde{\phi}_q = \gamma_q/\alpha_q^2. \tag{4.8}$$

Note that we know the exact value of γ_q and α_q.

In the light of the above equations, it is apparent that the update will never produce weight ϕ_j smaller than unity. Indeed, from equation (4.7) it follows that

$$\begin{aligned}\tilde{\phi}_j &\geq \tilde{\alpha}_j^2\alpha_q^2 + 1 - 2\tilde{\alpha}_j^2\alpha_q^2 + \tilde{\alpha}_j^2\gamma_q\\ &\geq \tilde{\alpha}_j^2\alpha_q^2 + 1 - 2\tilde{\alpha}_j^2\alpha_q^2 + \tilde{\alpha}_j^2(\alpha_q^2 + 1)\\ &= \tilde{\alpha}_j^2 + 1.\end{aligned}$$

The method of weight updates proposed above is clearly just a rough approximation of steepest edge pricing. It has the advantage of eliminating the single most expensive phase of update of exact weights. The results of numerical experiments (presented in Section 7.) have shown that in most cases it compares favourably with the Dantzig's pricing method both in terms of computation time and the number of iterations. It also seems to be more efficient than the steepest edge strategy.

5. New features

Development of a decomposition type method (see Ruszczyński [13], [14] and Świętanowski [16]) based on the primal simplex algorithm as well as experiments with new parallel approaches to linear programming (see Wierzbicki [20]) triggered emergence of a few new ideas concering the revised simplex method itself. A fresh look at the use of penalties in objective function instead of a two-phase method or composite objective, a notion of semi-basic solution and separation of solution vector and the basis are the most prominent new features of the implementation resulting from this research.

5.1 Non-basic variables no longer glued to their bounds

We have already shown in Section 2. that the simplex method can easily be generalized to accept the non-basic variables that are not fixed on their bounds. This relatively minor change has several interesting consequences. It is now possible to start the simplex algorithm with any $\mathbf{x}_N$ satisfying box constraints $\mathbf{l}_N \leq \mathbf{x}_N \leq \mathbf{u}_N$. Of other interesting consequences we will now only mention the impact this may have on degeneracy.

Definition 1. (semi-basic feasible solution) *A vector*

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix}$$

such that

$$\begin{gathered} [\mathrm{B} \ \ \mathrm{N}] \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix} = \mathbf{b}, \\ \mathbf{l}_B \leq \mathbf{x}_B \leq \mathbf{u}_B, \\ \mathbf{l}_N \leq \mathbf{x}_N \leq \mathbf{u}_N \end{gathered}$$

where

$$\mathbf{A} = [\mathrm{B} \ \ \mathrm{N}]$$

and B *is non-singular, is a semi-basic feasible solution of a linear problem (1.1)–(1.2).*

Recall that a degenerate iteration occurs when the out-going p-th basic variable is already on its finite bound before the step is made. The direction of the step η pushes the basic variable to its bound. Thus only a step of zero length may be made and q-th non-basic variable replaces the p-th in the basis. No progress (measured by objective function value decrease) is achieved. In standard simplex method the incoming variable is necessarily on it's bound and thus the degeneracy level (the number of basic variables which are on their bounds) remains unchanged.

Let us now assume that some of the non-basic variables are between their bounds. In such situation a degenerate step would still be performed, but if the introduced q-th variable was between its bounds rather than on one of them, then the degeneracy level would decrease and chances of a non-zero step in the next iteration would increase. Numerical test results from experiments with the most degenerate problems of the Netlib test collection proved this reasoning to be right. In particular, some difficult and highly degenerate problems could only be solved when using this technique.

Dropping the requirement that the non-basic variables always be on their bounds has many other simple yet, perhaps even more interesting consequences which will be enumerated in the following sections.

5.2 Search for an initial feasible solution

The penalty method for finding initial feasible solution (sometimes called "the big M method") has often been unduly criticized (see e.g. Nazareth [11]). The critics pointed out the difficulty of finding appropriate value of the penalty term M. If M should be too large in comparison with other non-zero entries of the objective vector $\mathbf{c}$, then numerical difficulties would occur. Other non-zeros in $\mathbf{c}$ would appear as insignificant disturbances when compared to the penalty term. A problem which has little to do with the original one would be solved. On the other hand, too small a value of M would allow the algorithm to produce an infeasible solution. In case of an ill-formulated problem which is actually infeasible it would be difficult to detect this infeasibility.

Indeed, the attempts at assessing *a priori* the value of penalty M by rough estimation of spectral norms of all possible simplex bases are bound to produce huge, and thus impractical values of M. We claim that in our approach in which penalty M is dynamically adjusted all the above mentioned difficulties have been successfully dealt with. We solve a problem with a well-scaled objective function vector. Throughout almost all iterations the objective does not have to be changed, which is advantageous whenever reduced costs' or dual variables' update scheme is employed. We propose a method for precise calculation of penalty M as well as a criterion for determining problem's actual infeasibility.

What is important, the algorithmic overhead imposed by dynamic control of the penalty value has proven in the series of tests to be negligible (see Section 7.).

5.2.1 Problem reformulation and initial feasible solution

Let us now restate the linear problem (1.1)–(1.2). We are concerned with minimizing

$$\min \mathbf{c}^T\mathbf{x} \tag{5.1}$$

subject to constraints

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{l} \leq &\mathbf{x} \leq \mathbf{u} \end{aligned} \tag{5.2}$$

where $\mathbf{l},\ \mathbf{u} \in \overline{\Re}^n$, $\overline{\Re} = \Re \cup \{-\infty, +\infty\}$. Additionally let us define index sets I_L and I_U as $I_L = \{i : l_i > -\infty\}$ and $I_U = \{i : u_i < +\infty\}$ respectively.

In order to be able to use easily any starting point $\mathbf{x}^0$, $\mathbf{l} \leq \mathbf{x}^0 \leq \mathbf{u}$ we reformulate the problem (5.1)–(5.2) by adding a vector of non-negative artificial variables $\mathbf{t}$:

$$\min \mathbf{c}^T\mathbf{x} + \mathbf{p}^T\mathbf{t} \tag{5.3}$$

s. t.

$$\begin{aligned} \mathbf{A}\mathbf{x} + \mathbf{J}\mathbf{t} &= \mathbf{b} \\ \mathbf{l} \leq \mathbf{x} &\leq \mathbf{u} \\ \mathbf{t} &\geq \mathbf{0} \end{aligned} \tag{5.4}$$

where $\mathbf{J} \in \Re^{m\times m}$ is a diagonal matrix such that $\mathbf{J}_{i,i} \in \{-1, +1\}$, $i = 1, \ldots, m$ and vector $\mathbf{p}$, $\mathbf{p} \in \Re^m$ is a penalty term

$$\mathbf{p} = \begin{bmatrix} M \\ \vdots \\ M \end{bmatrix}$$

in which $M > 0$.

The problem (5.3)–(5.4) has a feasible solution

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{t} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^0 \\ \mathbf{t}^0 \end{bmatrix}$$

where $\mathbf{t}^0$ is a vector representing the $\mathbf{x}^0$'s infeasibility in terms of constraints (5.2) of the original problem:

$$\mathbf{t}_i^0 = |(\mathbf{b} - \mathbf{A}\mathbf{x}^0)^T\mathbf{e}_i|$$

and

$$\mathbf{J}_{i,i} = \begin{cases} +1 & (\mathbf{b} - \mathbf{A}\mathbf{x}^0)^T \mathbf{e}_i \geq 0 \\ -1 & (\mathbf{b} - \mathbf{A}\mathbf{x}^0)^T \mathbf{e}_i < 0 \end{cases}, \quad i = 1, \ldots, m.$$

5.2.2 The pair of dual problems

Let us state dual problems for both (5.1)-(5.2) and (5.3)-(5.4) as

$$\max \mathbf{b}^T\mathbf{y} + \sum_{i \in I_L} w_i l_i - \sum_{i \in I_U} v_i u_i \tag{5.5}$$

subject to

$$\begin{array}{c} \mathbf{A}^T\mathbf{y} + \mathbf{w} - \mathbf{v} = \mathbf{c} \\ \mathbf{w}, \mathbf{v} \geq \mathbf{0} \\ w_i = 0 \text{ for } i \notin I_L \\ v_i = 0 \text{ for } i \notin I_U. \end{array} \tag{5.6}$$

and

$$\max \mathbf{b}^T\mathbf{y} + \sum_{i \in I_L} w_i l_i - \sum_{i \in I_U} v_i u_i \tag{5.7}$$

subject to

$$\begin{array}{c} \mathbf{A}^T\mathbf{y} + \mathbf{w} - \mathbf{v} = \mathbf{c} \\ \mathbf{J}\mathbf{y} \leq \mathbf{p} \\ \mathbf{w}, \mathbf{v} \geq \mathbf{0} \\ w_i = 0 \text{ for } i \notin I_L \\ v_i = 0 \text{ for } i \notin I_U \end{array} \tag{5.8}$$

respectively.

This re-statement of the problem pair offers one interesting insight into the real meaning of the penalty method (also commonly known as the "big M" method). Penalizing artificial variables which represent infeasibility is equivalent to imposing an upper bound of M on the dual variables. This effectively means that if fulfilling i-th row constraint of the original problem should cost us M or more per unit violation, then the constraint does not have to be satisfied.

5.2.3 Some properties of the reformulated problem

It is quite obvious that the reformulated problem (5.3)-(5.4) is either unbounded or has an optimal solution. We will not be concerned with the first case — unboundedness. Let us only note that the reformulated problem is a relaxation of the original one, and as such is unbounded if the original one is. This property may easily be proven.

Observation 5..1 (preservation of unboundedness) *If the original linear problem (5.1)-(5.2) is unbounded then so is the reformulated problem (5.3)-(5.4).*

Proof: Unboundedness of (5.1)-(5.2) means that there exists a feasible solution $\mathbf{x}^0$ and an extreme ray $\mathbf{d}$ such that:

- $\begin{cases} d_i \geq 0 & i \in I_L \\ d_i \leq 0 & i \in I_U \\ d_i \in \Re & \textit{otherwise} \end{cases}$
- $\mathbf{A}\mathbf{d} = \mathbf{0}$ and
- $\mathbf{c}^T\mathbf{d} < 0$.

Therefore for every $\varepsilon > 0$ $\mathbf{x}^0 + \varepsilon\mathbf{d}$ is a feasible solution to problem (5.1)–(5.2). Furthermore, for $\varepsilon \to +\infty$

$$\mathbf{c}^T\mathbf{x} = \mathbf{c}^T(\mathbf{x}^0 + \varepsilon\mathbf{d}) \to -\infty.$$

Clearly a pair $(\mathbf{x}^0, \mathbf{0})$ is then a feasible solution to problem (5.3)–(5.4) and there exists a ray $\overline{\mathbf{d}} = (\mathbf{d}, \mathbf{0})$ for which:

- $\begin{cases} \overline{d}_i \geq 0 & i \in I_L \cup \{n+1, \ldots, n+m\} \\ \overline{d}_i \leq 0 & i \in I_U \\ \overline{d}_i \in \Re & otherwise \end{cases}$
- $[\mathbf{A}\ \mathbf{J}] \begin{bmatrix} \mathbf{d} \\ \mathbf{0} \end{bmatrix} = \mathbf{Ad} = \mathbf{0}$ and
- $\begin{bmatrix} \mathbf{c}^T & \mathbf{p}^T \end{bmatrix} \begin{bmatrix} \mathbf{d} \\ \mathbf{0} \end{bmatrix} = \mathbf{c}^T\mathbf{d} < 0.$

Thus the reformulated problem is unbounded as well. ∎

Lemma 1. (optimal solution equivalence) *If the original problem (5.1)–(5.2) is not unbounded and has an optimal solution* $\hat{\mathbf{x}}$ *then there exists a finite positive number* M_0 *such that for every* $M \geq M_0$ *vector* $(\hat{\mathbf{x}}, \mathbf{0})$ *is an optimal solution to problem (5.3)–(5.4).*

Proof: Let us consider the optimal solution $\hat{\mathbf{y}}$ to original problem's dual (5.5)–(5.6). Existence of finite $\hat{\mathbf{x}}$ implies existence and finiteness of $\hat{\mathbf{y}}$. Let y_{max} denote the largest optimal dual variable

$$y_{max} = \max_{i=1,\ldots,m} \hat{y}_i.$$

A pair $(\hat{\mathbf{x}}, \mathbf{0})$ is a feasible primal solution to the reformulated problem (5.3)–(5.4). Let $M_0 = y_{max}$ and $M = M_0 + \varepsilon$, $\varepsilon > 0$. For such M $\hat{\mathbf{y}}$ defines an optimal solution to (5.7)–(5.8). Thus from the complementarity conditions for dual slack variables associated with (5.8) one gets

$$\hat{t}_i(M - \hat{y}_i) = 0.$$

The definition of M ensures $\hat{\mathbf{t}} = \mathbf{0}$. Consequently, the solution $(\hat{\mathbf{x}}, \mathbf{0})$ and $\hat{\mathbf{y}}$ is both primal and dual feasible (and optimal). ∎

The next observation specifies our requirements concerning the solution of the reformulated problem. If the original problem is feasible, we may demand that the artificial variables in the optimal solution to (5.3)–(5.4) all be equal to zero.

Observation 5..2 (optimal solution existence) *If the original linear problem (5.1)–(5.2) is not unbounded and has a feasible solution then there exists a finite positive number* M_0 *such that for every* $M \geq M_0$ *problem (5.3)–(5.4) has a feasible and optimal solution* $(\hat{\mathbf{x}}, \mathbf{0})$ *such that* $\hat{\mathbf{x}}$ *is an optimal solution to (5.1)–(5.2).*

Proof: If the original problem (5.1)–(5.2) is not unbounded and has a feasible solution then it also has a finite optimal solution. This and lemma 1. proves the observation. ∎

The following observation is the most important one. It shows (although indirectly) how infeasibility of the original problem can be proven. It is a direct consequence of observation 5..2.

Observation 5..3 (infeasibility detection) *If there does not exist a finite number* $M > 0$ *for which* $(\hat{\mathbf{x}}, \mathbf{0})$ *is an optimal solution of (5.3)–(5.4) then the original problem (5.1)–(5.2) is infeasible.*

Proof: Let us assume that (5.1)–(5.2) has a feasible solution $\mathbf{x}^0 > \mathbf{0}$. According to the observation 5..2 this implies existence of a finite M for which problem (5.3)–(5.4) has a feasible and optimal solution $(\hat{\mathbf{x}}, \mathbf{0})$ which contradicts the assumption. ∎

The theoretical results of this section may be summarized as follows:

1. if the original problem has a solution, we can find it by solving the reformulated problem with sufficiently large value of M,
2. we need to reduce the artificial variables to zero,
3. if we prove that it cannot be achieved by further increases of the value of M then we know the original problem is infeasible.

5.3 When is "big M" big enough?

When $\hat{\mathbf{t}} = \mathbf{0}$ then the reformulated problem is exactly equivalent to the original one and $\hat{\mathbf{x}}$ is the latter one's optimal solution. If, however, the original problem is infeasible, the reformulated one still has an optimal and feasible solution $(\hat{\mathbf{x}}, \hat{\mathbf{t}})$. The task of distinguishing between these two situations is the subject of this section. This problem may also be put differently: when is M big enough for us to be sure, that non-zero optimal value of $\mathbf{t}$ corresponds to the infeasibility of the original LP (5.1)–(5.2) (see also observation 5..2).

Note that the optimal basis of the problem (5.3)–(5.4) is also a feasible basis of its dual problem and a feasible solution to the above is also a feasible (but not necessarily optimal) solution to original problem's dual.

Let us assume that we have an optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{t}})$ to the reformulated problem (5.3)–(5.4) such that there exists i for which $t_i > 0$. From now on (as long as the optimality criterion is satisfied and artificial variables are not equal to zero) we will use a different pricing technique.

5.3.1 Split pricing

Constraints (5.8) may also be written as

$$\begin{aligned} \mathbf{A}^T\mathbf{y} + \mathbf{z_x} &= \mathbf{c} \\ \mathbf{I}\mathbf{y} + \mathbf{z_t} &= \mathbf{p} \\ \mathbf{z_x}, \mathbf{z_t} &\geq \mathbf{0} \end{aligned}$$

where $\mathbf{z_x} = \mathbf{w} - \mathbf{v}$ and $\mathbf{z_t}$ are reduced costs for the original and artificial variables respectively.

We now compute reduced cost for non-basic artificial variable t_i as

$$\begin{aligned} z_{\mathbf{t}i} &= M - \mathbf{y}^T\mathbf{e}_i = M - y_i && \text{for } J_{i,i} = +1 \text{ and} \\ z_{\mathbf{t}i} &= M - \mathbf{y}^T(-\mathbf{e}_i) = M + y_i && \text{for } J_{i,i} = -1 \end{aligned}$$

where $\mathbf{e}_i$ denotes i-th row (or column) of the identity martix $\mathbf{I}$.

Since we would like to decrease t_i from its current positive value (and possibly to zero), we want $z_{\mathbf{t}i}$ to be positive. In other words we want M to be greater than the current value of y_i. If there are no artificial variables in the basis then $\mathbf{y} = \mathbf{B}^{-T}\mathbf{c_B}$ does not depend on M and it is sufficient to increase M by $\varepsilon > 0$ to make the reduced cost positive.

It is, however, possible that $y_i = y_i(M)$. Then we have to find such M that would allow introduction of t_i into basis. In order to find out the dependency between $\mathbf{y}$ and M we split the basic cost vector $\mathbf{c_B}$ into two parts: $\mathbf{c_{Bx}}$ and $\mathbf{c_{Bt}}$ corresponding to variables $\mathbf{x}$ and $\mathbf{t}$ respectively.

We have

$$\mathbf{c_B} = \begin{bmatrix} \mathbf{c_{Bx}} \\ \mathbf{c_{Bt}} \end{bmatrix} = \begin{bmatrix} \mathbf{c_{Bx}} \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{c_{Bt}} \end{bmatrix}$$

and

$$\mathbf{y_x} = \mathbf{B}^{-T}\begin{bmatrix} \mathbf{c_{Bx}} \\ \mathbf{0} \end{bmatrix}, \mathbf{y_t} = \mathbf{B}^{-T}\begin{bmatrix} \mathbf{0} \\ \mathbf{c_{Bt}} \end{bmatrix}.$$

Since

$$\mathbf{c_{Bt}} = \begin{bmatrix} M \\ \vdots \\ M \end{bmatrix} = M\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = M\mathbf{e}$$

we may extract the subgradient $\overline{\mathbf{y_t}}$ of $\mathbf{y_t}$

$$\mathbf{y_t} = M\left(\mathbf{B}^{-T}\begin{bmatrix} \mathbf{0} \\ \mathbf{e} \end{bmatrix}\right) = M\overline{\mathbf{y_t}}.$$

Note that $\overline{\mathbf{y_t}}$ is simply a sum of those rows of the basis inverse, to which the basic artificial variables correspond.

It goes without saying that

$$\mathbf{y} = \mathbf{y_x} + \mathbf{y_t} = \mathbf{y_x} + M\overline{\mathbf{y_t}}.$$

And so we conclude, that

$$\begin{array}{ll} z_{\mathbf{t}i} = M - \mathbf{y_x}_i - M\overline{\mathbf{y_t}}_i & \text{for } J_{i,i} = +1 \text{ and} \\ z_{\mathbf{t}i} = M + \mathbf{y_x}_i + M\overline{\mathbf{y_t}}_i & \text{for } J_{i,i} = -1. \end{array}$$

5.3.2 Dynamic penalty control criteria

We need $z_{\mathbf{t}i}$ to become non-zero. We shall distinguish the following cases:

1. if $J_{i,i} = +1$ then
   - if $\overline{\mathbf{y_t}}_i = 1$ then the value of $z_{\mathbf{t}i}$ is independent of M, so an increase of M can not produce a non-zero reduced cost,
   - if $\overline{\mathbf{y_t}}_i < 1$ then an increase of M so that
   $$M > \frac{\mathbf{y_x}_i}{1 - \overline{\mathbf{y_t}}_i}$$
   will result in positive reduced cost $z_{\mathbf{t}i}$ and possibly with decrease of t_i in the subsequent simplex iteration,
   - if $\overline{\mathbf{y_t}}_i > 1$ then
   $$\begin{aligned} z_{\mathbf{t}i} &= M - \mathbf{y_x}_i - M\overline{\mathbf{y_t}}_i \\ &= M(1 - \overline{\mathbf{y_t}}_i) - \mathbf{y_x}_i \end{aligned}$$
   and obviously $\mathbf{y_x}_i$ is already negative:
   $$\mathbf{y_x}_i = M(1 - \overline{\mathbf{y_t}}_i) < 0.$$
   This means that if we increase M it will make *increasing* t_i profitable. We use the same formula as before:
   $$M > \frac{\mathbf{y_x}_i}{1 - \overline{\mathbf{y_t}}_i}$$
2. if $J_{i,i} = +1$ then
   - if $\overline{\mathbf{y_t}}_i = -1$ then $z_{\mathbf{t}i}$ is independent of M,
   - otherwise we state that penalty M should be greater than
   $$M > \frac{-\mathbf{y_x}_i}{1 + \overline{\mathbf{y_t}}_i}.$$

Similarly for variable x_j the reduced cost $z_{\mathbf{x}j}$ (which we want to become non-zero) is computed as

$$z_{\mathbf{x}j} = c_j - (\mathbf{y_x} + M\overline{\mathbf{y_t}})^T \mathbf{a}_j.$$

Depending on the bound on which the non-basic structural variable x_j is we decide on further action:

1. if x_j is at its finite lower bound and $\overline{\mathbf{y_t}}^T \mathbf{a}_j > 0$ or
2. if x_j is at its finite upper bound and $\overline{\mathbf{y_t}}^T \mathbf{a}_j < 0$ or
3. if x_j is between its bounds and $\overline{\mathbf{y_t}}^T \mathbf{a}_j \neq 0$

we demand that

$$M > \frac{c_j - \mathbf{y_x}^T \mathbf{a}_j}{\overline{\mathbf{y_t}}^T \mathbf{a}_j}.$$

Note that since for any linear programming problem there is only a finite number of possible simplex bases, it is in principle (but not in practice) possible to compute appropriately big value of M (by search of all possible simplex bases) or detect problem's infeasibility without actually solving it. This is a direct proof of observation 5..3.

5.3.3 Required modifications to standard pricing

Some applications in decomposition schemes (see [16]) require all non-zero artificial variables present in the optimal solution to the modified problem to be in the optimal basis. To this end we need to modify slightly the pricing method used in the primal simplex algorithm.

Typically we only consider variables with reduced cost z_j which guarantees a minimum profit of δ_O per unit change of non-basic variable x_j, where $\delta_O > 0$ is called optimality tolerance.[6] In our case we want all non-zero artificial variables to be present in the optimal basis. In case of artificial variables we treat reduced costs of zero as profitable. Of course, if some other variables have non-zero and profitable reduced costs, they are chosen as candidates to enter the basis.

The result of this procedure is such that

- the algorithm introduces into the basis the same columns as it would otherwise, until no more variables have favourable reduced costs and
- when some non-zero artificial variables remain outside of basis in the optimum, thay are "forced" into the basis.

5.3.4 Algorithm's expected behaviour

In case of a feasible problem (5.1)–(5.2) we are only able to predict that all artificial variables will be reduced to zero. Some of them may be in the optimal basis, some may be not. But this is not the most interesting case.

We are more concerned with a solution process of an infeasible problem. We expect that after the reformulated problem (5.3)–(5.4) is solved and some artificial variables are found to be non-zero, the penalty M will be increased (unless it s found to be pointless — according to the formulas presented before). The algorithm will then tend to put more and more non-zero artificial variables into the basis. In general (especially in case of infeasible problems) we must take into account presence of artificial variables in the final (optimal) basis of the reformulated problem (5.3)–(5.4).

As it was mentioned before, if we initially assign a very large value to M we are likely to cause unnecessary numerical difficulties as soon as the first artificial variable is introduced into

[6] In the steepest edge algorithm we compare objective function value decrease per unit move along the edge of the simplex.

the basis. If M is very large (compared to other cost vector coefficients of the basic variables), it will dominate the shadow prices (dual variables) used during pricing. Since we now know exactly "how big should the big M be", we may start the algorithm with a relatively small penalty M (e.g. $M = \rho \max_j c_j$, $\rho \geq 1$) and increase it when necessary to e.g. ten times the minimum value that would allow us to make a step. We recommend using at least a factor of two in order to avoid unnecessarily many adjustments to the penalty term. Indeed, our experience (see Section 7.) shows that $\rho = 2$ is perfectly satisfactory. Note that in case of dual variable or reduced cost update methods, the updated vectors would have to be computed afresh after each change of the penalty factor M.

5.4 A heuristic for infeasibility reduction

The method for finding a feasible initial solution presented in this paper may in general produce a dense residual vector

$$\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0$$

before the artficial variables are added. We developed a simple yet efficient heuristic which (whenever possible) tries to shift variables x_j from their initial positions x_j^0, $j = 1, \ldots, n$ so that the residual vector is reduced. We want to decrease a number of non-zero residuals r_j^K as well as Euclidean norm of the vector $\mathbf{r}^K$, where K denotes the last step of the algorithm — the one in which the reduction stops.

We scan the whole constraint matrix c_1 times, $c_1 > 0$ or until $||\mathbf{r}^k||$ reaches zero. In each pass we change only one variable x_j, $j = 1, \ldots, n$ at a time so that

(a) the norm of the current residual vector $\mathbf{r}^k$ decreases and

(b) no more than c_2, $c_2 \geq 0$ new non-zeros appear in the residual vector.

For every variable x_j for which its corresponding column $\mathbf{a}_j$ has no more than c_2 non-zeros at positions at which $\mathbf{r}^k$ has zero entries, we solve a simple optimization problem:

$$\min_{l_j \leq x_j \leq u_j} d(x_j^k)$$

where

$$\begin{aligned} d(x_j^k) &= ||\mathbf{r}^k - \mathbf{a}_j^T x_j^k||^2 \\ &= \mathbf{a}_j^T \mathbf{a}_j (x_j^k)^2 - 2\mathbf{a}_j^T \mathbf{r}^k x_j^k + (\mathbf{r}^k)^T \mathbf{r}^k. \end{aligned}$$

Clearly $d(x_j^k)$ is a convex quadratic function of variable x_j^k. It reaches its global minimum at

$$\tilde{x}_j^k = \frac{\mathbf{a}_j^T \mathbf{r}^k}{\mathbf{a}_j^T \mathbf{a}_j}.$$

If $\tilde{x}_j^k$ should be infeasible, we project it onto interval $\langle l_j, u_j \rangle$:

$$x_j^{k+1} = \begin{cases} \tilde{x}_j^k & l_j \leq \tilde{x}_j^k \leq u_j \\ l_j & \tilde{x}_j^k < l_j \\ u_j & \tilde{x}_j^k > u_j. \end{cases}$$

If $x_j^{k+1} \neq x_j^k$ then

$$\mathbf{r}^{k+1} = \mathbf{r}^k - \mathbf{a}_j (x_j^{k+1} - x_j^k),$$

otherwise $\mathbf{r}^{k+1} = \mathbf{r}^k$. This ends k-th step; k is incremented, j is moved to point to the next column (or the first one if $j = n$) and the algorithm continues until there are no more residuals or c_1 passes have been completed.

The computational effort required is rather moderate. We could rely entirely on updates of the residual vector, but at the beginning of each pass we compute it afresh. In this manner we prevent excessive build-up of round-off errors. We need to compute the norms of all columns of matrix $\mathbf{A}$ only once. In every step k we calculate a scalar product $\mathbf{a}_j^T\mathbf{r}^k$. It is inexpensive because $\mathbf{a}_j$ is a sparse vector (typically it has no more than 5 to 10 entries).

Note that it is possible that after the initial basis is chosen, some of the non-basic variables will be between their bounds. As it was explained before this is advantageous, because it helps to avoid performing degenerate iterations.

5.5 Consequences for the simplex method

The whole Section 5. has been devoted to the description of a new version of the revised simplex algorithm. We reiterate the consequences our approach has for the method:

1. The basis and solution are now two seperate entities no longer bound together into a "basic solution". This allows us to start (or restart) the algorithm with any combination of a feasible solution $\mathbf{x}$ and a non-singular basis $\mathbf{B}$.

2. Any point $\mathbf{x} \in \Re^n$ is a legitimate starting point. The penalty method will take care of $\mathbf{x}$'s infeasibility. Of course very large values of some x_j will cause huge residuals and later numerical difficulties. But so would any other unnecessarily large numbers (e.g. constraint matrix coefficients) in the formulation of the problem.

3. An inexpensive tool allowing to reduce degeneration was proposed.

In the future some features of our implementation will allow us to try even more interesting approaches to linear programming. For example it might become possible to perform non-simplex steps, i.e. steps not along the edges of the simplex, but across its facets or even through the interior. Methods for finding non-simplex directions are beyond the scope of this paper (but certainly not beyond imagination).

6. Restart of the penalty based simplex

Let us assume that an initial solution $\mathbf{x}^0$ and a non-singular initial basis $\mathbf{B}^0$ are given. The following algorithm may be used to start the penalty-based simplex from this solution and basis:

Algorithm III: Restart of the penalty-based primal simplex method

0. An initial solution $\mathbf{x}^0$ and a non-singular initial basis $\mathbf{B}^0$ are given.

1. The non-basic variables which violate their simple bounds are projected on those bounds:

$$\mathbf{x}_{N\,j} = \begin{cases} \mathbf{l}_{N\,j} & \mathbf{x}^0_{N\,j} < \mathbf{l}_{N\,j} \\ \mathbf{u}_{N\,j} & \mathbf{x}^0_{N\,j} > \mathbf{u}_{N\,j} \\ \mathbf{x}^0_{N\,j} & \text{in all other cases} \end{cases}$$

$$j = m+1, \ldots, n.$$

2. The basic variables are calculated:

$$\mathbf{x}_B = \mathbf{B}^{-1}(\mathbf{b} - \mathbf{N}\mathbf{x}_N).$$

3. The infeasible basic variables are projected on intervals delimited by their box constraints:

$$\mathbf{x}_{B\,j} = \begin{cases} \mathbf{l}_{B\,j} & \mathbf{x}_{B\,j} < \mathbf{l}_{B\,j} \\ \mathbf{u}_{B\,j} & \mathbf{x}_{B\,j} > \mathbf{u}_{B\,j} \\ \mathbf{x}^0_{B\,j} & \text{in all other cases.} \end{cases}$$

$$j = 1, \ldots, m$$

4. The residuals are computed:

$$\mathbf{r} = \mathbf{b} - \mathbf{N}\mathbf{x}_N - \mathbf{B}\mathbf{x}_B.$$

   If $\mathbf{r} = \mathbf{0}$ the algorithm terminates.

5. Residuals are decreased by means of shifting some variables from their current positions (see Section 5.4 for a description of a heuristic used). If the residuals are reduced to zero the restart algorithm terminates.

6. Artificial variables $\mathbf{t}$ are added to the problem in order to remove the remaining infeasibility. Their non-zero values are penalized. For details consult Section 5.2.

7. A linear problem to which a feasible solution is known has thus been formulated. The restart algorithm terminates.

One remark concerning the initial basis $\mathbf{B}^0$ is in order. As it has been stated in Section 5. it is in general possible that the final (perhaps optimal) basis will contain some artificial variables. When we use this basis as a starting basis for another problem, the same artificial variables have to be used again, even if the non-zeros in their corresponding columns have now different value.

7. Numerical results

The numerical results that are presented in this section will not surprise the reader. Perhaps the choice of test problems requires more explanation then the tables themselves. We used about 50 of the problems included in the NETLIB test LP collection of Gay [7], which is probably quite familiar to readers who encountered other papers concerned with linear programming implementations. An overview of about 90 of those problems (listing their dimensions, optimal solutions and other details) is presented in [2].

In our tests we used a subset of those 90 LP's. We rejected the smallest problems (those with no more than 2000 non-zeros in the constraint matrix). The reason for this omission is rather obvious: we are concerned with large and sparse linear problems and we think that the solution process of small (and consequently relatively dense) linear programs does not offer much insight into the possible performance of our code in real-life situations.

We present a number of tables; each demonstrates one of the features of our code. Table 1 shows the results of applying our infeasibility reduction heuristic to the initial solutions obtained from a basis creation technique similar to this of Bixby [2]. In Tables 2 and 3 a comparison of the performance of three pricing algorithms is given. Finally, dynamic penalty control is cofronted with large static penalty in Table 4. Unless otherwise noted, the code was run with dynamic penalty factor control and employed full pricing with reduced cost updates and approximate steepest edge calculation. All the computations were performed on the same 40MHz SPARC Station 2 computer. Computation times are measured in CPU seconds as reported by the Solaris operating system.

7.0.1 Infeasibility reduction heuristic

Table 1 presents the reports of the infeasibility reduction routine described in Section 5.4 with $c_1 = 2$ and $c_2 = 1$. That means that at most two passes of the constraint matrix were performed and only one non-zero creation in the residual vector per variable shift was allowed (see Section 5.4 for details). We have selected those numbers after some initial experiments which proved them to be reasonable, but not necessarily the best. It is likely that more "fine tuning" of those parameters would help. Each table row corresponds to one linear problem. First the name of the problem and the number of rows of its constraint matrix are given. Infeasibility before reduction (denoted by *Initial infeas.* in the table) is expressed by the number of non-zeros in the residual vector (*no.*) and the euclidean norm of this vector (*norm*). The same data after the reduction is presented under the heading *Final infeas.*. The number of reduction passes is specified in the next column (*Pass.*). Finally, infeasibility reduction in terms of the number of non-zeros of the residual vector (*Number red.*) as well as its norm (*Norm red.*) are expressed in percent and calculated as

$$\text{reduction} = \frac{\text{number\_after\_reduction} - \text{number\_before\_reduction}}{\text{number\_before\_reduction}} \cdot 100[\%].$$

Naturally, negative number corresponds to an *increase* in the number of non-zeros (the increase in the norm of the vector by our algorithm is not possible).

It is easy to see that the behaviour of the algorithm depends highly on the problem. We can see three major patterns:

- the initial solution found by crashing is feasible or
- the initial solution is not feasible, but the reduction routine is unable to reduce the residuals and terminates after one pass or
- the reduction progresses and two passes are completed.

On some smaller problems or with different values of the parameters c_1 and c_2 the reduction heuristic has sometimes managed to find a feasible primal solution, however in general this may be considered a rather rare occurence. In case of 6 problems the method increased the number of non-zeros. The infeasibility of 7 LP's remained unchanged (and therefore the reduction necessarily caused a nett loss of computation time). In general, though, most of the initial solutions benefited from applying the reduction algorithm. Additionally, it is worth remembering that this technique increases the number of non-basic variables placed between their bounds and thus may have beneficial impact on solution process of highly degenerate problems.

While the average gains produced by our algorithm are encouraging (30% decrease in the euclidean norm of the initial residual vector), we see that its application should be decided upon on a problem-by-problem basis.

7.0.2 Three pricing techniques

Papers by Goldfarb and Reid [8], Forrest and Goldfarb [5], Harris [9], Bixby [3] and many others have discussed the practicability, efficiency and implementation of different pricing techniques for the modified simplex method. The most important of those are

(i) full, partial and/or multiple pricing,

(ii) DEVEX pricing (currently recognized as an approximation to steepest edge),

(iii) steepest edge and

(iv) hybrid approaches (typically used in commercial codes because of efficiency considerations).

It is now common knowledge that (i) is cheapest per iteration (especially when implemented in a reasonable manner, e.g. with dual variables' or reduced costs' updates) and offers reasonable overall efficiency. On the other hand (iii) adds some extra workload to each simplex iteration, but usually results in a significant decrease in the number of iterations. It is especially well suited for numerically difficult and highly degenerate problems, where the simpler methods may fail. DEVEX is one of the possible compromises between the former and the latter. In terms of computation time it seems superior both to steepest edge and minimum reduced cost pricing criteria. Finally, so-called hybrid approaches employ two or more of the basic techniques and switch between them when it seems that the solution process might benefit (see e.g. Bixby [3] for a description of such a hybrid method).

Our results gathered in Tables 2 and 3 conform to the findings of the above mentioned papers. In the tables we present the number of iterations (*Iter.*) needed to solve each problem with each of the three methods implemented in our code and the CPU time taken (*Time*). The compared methods are:

- most negative reduced cost criterion of Dantzig (*Reduced Cost* or *RC*),
- steepest edge (also denoted by *SE*) and
- our approximate steepest edge (*Approx. SE* and *ASE*).

While Table 2 lists the numbers of iterations and CPU times, Table 3 compares the methods by listing improvements in terms of solution times as well as the numbers of iterations measured in percent.

Steepest edge compared to the most negative reduced cost criterion almost always reduces the number of iterations (on the average by 15%) and quite as often increases the solution time (34%). Approximate steepest edge cuts down the average of 17% of iterations and 13% of time needed by the Dantzig's method. Both those results come as no surprise. What is quite interesting though, is the fact that approximate steepest edge is usually as good in terms of the number of iterations as the exact version! And, naturally, it is much faster.

7.0.3 Dynamic penalty control: efficiency and numerical stability

We shall present the impact of the dynamic penalty control method derived in Section 5. on the solution times and numbers of iterations of the simplex method employing Dantzig's pricing strategy. The comparison is shown in Table 4. As previously, the number of iterations and problem solution times are shown. Then the improvement offered by dynamic penalty control is computed (under the heading *Comparison*). Additionally, for dynamic penalty method the number of necessary penalty factor adjustments is given (*Pen. adj.*). In this experiment we used the initial dynamic penalty factor M computed as

$$M = 2 \max_{j=1,\dots,n} c_j.$$

Dynamic penalty method is definitely not a performance booster. In fact, on two problems, namely SCSD6 and SCSD8 it performed *very* badly. In most cases it reduces solution time by a tiny fraction, but sometimes the improvement reaches over thirty percent. The avarage improvement is negligibly small (and remains small, but positive if we disregard the two LP's mentioned above). What the table does not show is the impact of the dynamic penalty method on numerically difficult problems. The poorer is the scaling of the problem's objective function, the more disastrous effects may the large static penalty method have. We have observed that the dynamic penalty tends to solve numerically difficult problems, for which the static penalty method fails.

Now just a word about the penalty adjustments. From the description given in Section 5.3 it might seem that the penalty adjustments are a rather costly operation. When standard pricing discovers the need for penalty adjustment, up to two linear systems with basis transpose have to be solved, and then a "duplicate" pricing step is performed. The reader might be discouraged by possible excessive costs of such operations. The experiment proved those fears to be unsubstantiated. Only six problems needed penalty adjustments and it was always needed only once. In other experiments (with different initial penalty values or with other linear problems) we never observed more than two penalty adjustments.

The method may be thus seen as an inexpensive way of making the penalty method work without causing numerical difficulties. The obvious advantage of penalty method is the fact that it does not require a two phase simplex algorithm (in the first phase we cannot perform steepest edge pricing or reduced cost updates). It may be viewed as an alternative to a two-phase algorithms.

It has one additional feature that may be seen to be an advantage or a problem — depending on one's standpoint. The solver produces a solution even when the problem is not feasible. It may be seen as a waste of precious computation time or as a way of obtaining interestig information about the problem. The optimal solution to the modified problem may be used to reformulate the infeasible LP. And after necessary problem modifications the solution process may be restarted from the point in which the infeasibility was discovered.[7]

[7]This feature was already successfully tried in a decomposition scheme, but its description is far beyond the scope of this work.

Table 1: A test of the infeasibility reducing heuristic

Problem	Rows	Initial infeas.		Final infeas.		Pass.	Number red.	Norm red.
name		no.	norm	no.	norm		[%]	[%]
agg2	517	6	$2.295E+4$	3	$1.970E+4$	2	50.00	14.16
agg3	517	1	$1.177E+5$	4	$3.757E+4$	2	−300.00	68.08
bandm	306	1	$1.947E+3$	6	$1.760E+3$	2	−500.00	9.60
beaconfd	174	1	$5.670E+3$	1	$2.048E+3$	2	0.00	63.88
bnl1	644	57	$1.878E+3$	52	$1.531E+3$	2	8.77	18.48
boeing1	352	10	$2.012E+3$	10	$2.012E+3$	1	0.00	0.00
brandy	221	95	$6.035E+2$	77	$5.550E+2$	2	18.95	8.04
czprob	930	37	$7.467E+4$	14	$4.071E+4$	2	62.16	45.48
degen3	1504	11	$9.059E+1$	11	$9.059E+1$	1	0.00	0.00
e226	224	38	$9.612E+1$	31	$9.609E+1$	2	18.42	0.03
etamacro	401	13	$2.109E+2$	73	$2.037E+2$	2	−461.54	3.41
fffff800	525	14	$7.217E+5$	11	$1.072E+5$	2	21.43	85.15
finnis	498	10	$1.359E+4$	96	$7.269E+3$	2	−860.00	46.51
fit1d	25	0	$0.000E+0$	0	$0.000E+0$	0	0.00	0.00
fit1p	628	0	$0.000E+0$	0	$0.000E+0$	0	0.00	0.00
fit2d	26	0	$0.000E+0$	0	$0.000E+0$	0	0.00	0.00
fit2p	3001	0	$0.000E+0$	0	$0.000E+0$	0	0.00	0.00
forplan	162	53	$3.418E+4$	28	$5.150E+3$	2	47.17	84.93
ganges	1310	0	$0.000E+0$	0	$0.000E+0$	0	0.00	0.00
gfrd-pnc	617	2	$9.809E+4$	2	$9.809E+4$	1	0.00	0.00
grow7	141	0	$0.000E+0$	0	$0.000E+0$	0	0.00	0.00
israel	175	8	$2.155E+3$	8	$2.155E+3$	1	0.00	0.00
nesm	663	33	$5.243E+3$	21	$3.493E+3$	2	36.36	33.38
pilot4	411	80	$3.307E+3$	72	$3.162E+3$	2	10.00	4.38
scfxm1	331	11	$5.901E+3$	69	$4.028E+3$	2	−527.27	31.74
scfxm2	661	23	$8.522E+3$	14	$5.927E+3$	2	39.13	30.45
scfxm3	991	35	$1.140E+4$	21	$7.929E+3$	2	40.00	30.45
scrs8	491	54	$1.808E+1$	52	$1.542E+1$	2	3.70	14.71
scsd1	78	11	$1.129E+1$	11	$1.417E+0$	2	0.00	87.45
scsd6	148	13	$5.637E+1$	13	$6.244E+0$	2	0.00	88.92
scsd8	398	11	$1.707E+2$	11	$3.682E+1$	2	0.00	78.43
sctap2	1091	51	$1.205E+2$	51	$1.205E+2$	1	0.00	0.00
sctap3	1481	62	$1.044E+2$	62	$1.044E+2$	1	0.00	0.00
seba	516	18	$4.113E+2$	9	$4.004E+2$	2	50.00	2.65
shell	537	33	$3.709E+5$	33	$3.276E+5$	2	0.00	11.67
ship04l	403	11	$5.532E+1$	6	$2.087E+1$	2	45.45	62.27
ship04s	403	19	$4.952E+1$	11	$1.552E+1$	2	42.11	68.66
ship08l	779	13	$5.940E+1$	8	$5.571E+1$	2	38.46	6.21
ship08s	779	30	$4.374E+1$	18	$3.968E+1$	2	40.00	9.28
ship12l	1152	65	$1.326E+2$	32	$2.146E+1$	2	50.77	83.82
ship12s	1152	67	$1.035E+2$	33	$3.566E+1$	2	50.75	65.55
sierra	1228	25	$1.018E+4$	17	$9.182E+3$	2	32.00	9.80
standata	360	14	$2.415E+2$	11	$1.046E+2$	2	21.43	56.69
standgub	362	14	$2.415E+2$	11	$1.046E+2$	2	21.43	56.69
standmps	468	39	$3.320E+2$	17	$1.066E+2$	2	56.41	67.89
stocfor2	2158	34	$6.755E+3$	16	$2.670E+3$	2	52.94	60.47
tuff	334	2	$9.627E+0$	2	$9.627E+0$	1	0.00	0.00
woodw	1099	1	$3.217E+2$	9	$2.084E+2$	2	−800.00	35.22
80bau3b	2263	6	$7.558E+3$	4	$3.564E+3$	2	33.33	52.84
Average:						1.61	−52.20	30.56

Table 2: Pricing methods — absolute performance

Problem	Reduced Cost		Steepest Edge		Approx. SE	
name	Iter.	Time	Iter.	Time	Iter.	Time
agg2	180	2.7	202	4.7	208	3.3
agg3	211	3.4	217	5.3	216	3.7
bandm	771	15.0	447	14.6	459	8.8
beaconfd	47	0.3	39	0.4	39	0.2
bnl1	2725	99.8	1970	122.6	1725	70.7
boeing1	748	11.7	697	18.6	651	10.6
brandy	388	6.7	324	8.8	343	5.2
czprob	1286	58.4	996	84.1	1108	54.2
degen3	11739	1190.9	5067	932.6	4104	372.2
e226	691	11.0	416	10.2	370	5.8
etamacro	1131	20.7	717	19.2	729	13.0
fffff800	1305	35.9	890	41.9	759	20.0
finnis	903	15.6	676	17.6	675	12.5
fit1d	2079	63.3	1363	78.9	1279	37.5
fit1p	842	41.0	539	52.4	790	35.9
fit2d	39695	14409.1	20761	12263.5	21345	6356.5
fit2p	10144	5330.9	9974	5342.4	9672	2904.3
forplan	422	7.3	288	8.5	248	4.5
ganges	441	14.0	477	27.6	404	13.0
gfrd-pnc	517	8.1	512	16.0	540	9.9
grow7	178	4.8	203	8.4	206	5.6
israel	287	4.1	220	5.5	184	2.7
nesm	4652	242.1	3605	262.4	2808	141.4
pilot4	1252	59.7	1153	88.3	983	55.2
scfxm1	463	6.2	377	8.1	373	5.2
scfxm2	934	22.9	775	31.0	800	20.2
scfxm3	1445	52.1	1305	72.3	1254	47.6
scrs8	561	14.0	427	15.9	477	13.5
scsd1	460	5.0	223	4.1	317	4.3
scsd6	1224	21.1	895	28.4	668	12.6
scsd8	3048	116.3	2220	148.4	1279	47.1
sctap2	872	33.3	995	49.0	843	33.7
sctap3	1251	62.8	1271	83.8	1032	51.3
seba	222	1.3	218	1.8	218	1.5
shell	276	5.7	318	10.3	346	7.8
ship04l	236	4.9	242	7.1	239	5.4
ship04s	185	3.2	178	4.4	178	3.1
ship08l	487	21.4	499	34.1	539	26.8
ship08s	304	9.5	299	14.1	324	11.3
ship12l	913	52.3	901	72.8	907	59.0
ship12s	484	19.0	485	28.3	485	20.1
sierra	767	30.1	691	34.2	713	26.5
standata	86	1.4	79	1.9	90	1.6
standgub	86	1.4	79	2.0	90	1.7
standmps	265	4.5	274	8.2	257	4.6
stocfor2	1975	150.7	1644	188.2	1672	132.9
tuff	270	6.2	207	9.0	143	4.7
woodw	2040	210.1	1528	325.2	1474	171.3
80bau3b	8223	923.6	7053	1050.8	7140	837.3

Table 3: Pricing methods — relative performance

Problem	SE vs. RC		ASE vs. RC		ASE vs. SE
name	Iter.	Time	Iter.	Time	Iter.
agg2	−12.22	−74.07	−15.56	−22.22	−2.97
agg3	−2.84	−55.88	−2.37	−8.82	0.46
bandm	42.02	2.67	40.47	41.33	−2.68
beaconfd	17.02	−33.33	17.02	33.33	0.00
bnl1	27.71	−22.85	36.70	29.16	12.44
boeing1	6.82	−58.97	12.97	9.40	6.60
brandy	16.49	−31.34	11.60	22.39	−5.86
czprob	22.55	−44.01	13.84	7.19	−11.24
degen3	56.84	21.69	65.04	68.75	19.01
e226	39.80	7.27	46.45	47.27	11.06
etamacro	36.60	7.25	35.54	37.20	−1.67
fffff800	31.80	−16.71	41.84	44.29	14.72
finnis	25.14	−12.82	25.25	19.87	0.15
fit1d	34.44	−24.64	38.48	40.76	6.16
fit1p	35.99	−27.80	6.18	12.44	−46.57
fit2d	47.70	14.89	46.23	55.89	−2.81
fit2p	1.68	−0.22	4.65	45.52	3.03
forplan	31.75	−16.44	41.23	38.36	13.89
ganges	−8.16	−97.14	8.39	7.14	15.30
gfrd-pnc	0.97	−97.53	−4.45	−22.22	−5.47
grow7	−14.04	−75.00	−15.73	−16.67	−1.48
israel	23.34	−34.15	35.89	34.15	16.36
nesm	22.51	−8.38	39.64	41.59	22.11
pilot4	7.91	−47.91	21.49	7.54	14.74
scfxm1	18.57	−30.65	19.44	16.13	1.06
scfxm2	17.02	−35.37	14.35	11.79	−3.23
scfxm3	9.69	−38.77	13.22	8.64	3.91
scrs8	23.89	−13.57	14.97	3.57	−11.71
scsd1	51.52	18.00	31.09	14.00	−42.15
scsd6	26.88	−34.60	45.42	40.28	25.36
scsd8	27.17	−27.60	58.04	59.50	42.39
sctap2	−14.11	−47.15	3.33	−1.20	15.28
sctap3	−1.60	−33.44	17.51	18.31	18.80
seba	1.80	−38.46	1.80	−15.38	0.00
shell	−15.22	−80.70	−25.36	−36.84	−8.81
ship04l	−2.54	−44.90	−1.27	−10.20	1.24
ship04s	3.78	−37.50	3.78	3.13	0.00
ship08l	−2.46	−59.35	−10.68	−25.23	−8.02
ship08s	1.64	−48.42	−6.58	−18.95	−8.36
ship12l	1.31	−39.20	0.66	−12.81	−0.67
ship12s	−0.21	−48.95	−0.21	−5.79	0.00
sierra	9.91	−13.62	7.04	11.96	−3.18
standata	8.14	−35.71	−4.65	−14.29	−13.92
standgub	8.14	−42.86	−4.65	−21.43	−13.92
standmps	−3.40	−82.22	3.02	−2.22	6.20
stocfor2	16.76	−24.88	15.34	11.81	−1.70
tuff	23.33	−45.16	47.04	24.19	30.92
woodw	25.10	−54.78	27.75	18.47	3.53
80bau3b	14.23	−13.77	13.17	9.34	−1.23
Average:	15.13	−34.27	17.03	13.48	2.18

Table 4: Dynamic penalty control vs. large static penalty

Problem name	Static penalty Iter.	Static penalty Time	Dynamic penalty Iter.	Dynamic penalty Pen. adj.	Dynamic penalty Time	Comparison Iter. [%]	Comparison Time [%]
agg2	193	2.7	180	0	2.7	6.74	0.00
agg3	231	3.5	211	0	3.4	8.66	2.86
bandm	764	14.3	771	0	15.0	−0.92	−4.90
beaconfd	47	0.3	47	0	0.3	0.00	0.00
bnl1	2676	99.9	2725	1	99.8	−1.83	0.10
boeing1	742	10.7	748	0	11.7	−0.81	−9.35
brandy	468	7.6	388	0	6.7	17.09	11.84
czprob	1479	68.1	1286	0	58.4	13.05	14.24
degen3	14505	1671.2	11739	0	1190.9	19.07	28.74
e226	776	16.6	691	0	11.0	10.95	33.73
etamacro	1183	28.8	1131	0	20.7	4.40	28.12
fffff800	1163	46.1	1305	1	35.9	−12.21	22.13
finnis	815	19.6	903	0	15.6	−10.80	20.41
fit1d	2079	86.9	2079	0	63.3	0.00	27.16
fit1p	842	61.1	842	0	41.0	0.00	32.90
fit2d	39695	14401.0	39695	0	14401.0	0.00	0.00
fit2p	10144	5166.6	10144	0	5166.6	0.00	0.00
forplan	441	7.0	422	1	7.3	4.31	−4.29
ganges	439	13.5	441	0	14.0	−0.46	−3.70
gfrd-pnc	513	7.7	517	0	8.1	−0.78	−5.19
grow7	178	4.7	178	0	4.7	0.00	0.00
israel	287	4.0	287	0	4.0	0.00	0.00
nesm	4085	199.0	4652	1	242.1	−13.88	−21.66
pilot4	Not solved!		1252	0	59.7	—	—
scfxm1	454	5.9	463	0	6.2	−1.98	−5.08
scfxm2	989	23.5	934	0	22.9	5.56	2.55
scfxm3	1512	52.4	1445	0	52.1	4.43	0.57
scrs8	560	13.1	561	0	14.0	−0.18	−6.87
scsd1	441	4.8	460	0	5.0	−4.31	−4.17
scsd6	770	14.2	1224	1	21.1	−58.96	−48.59
scsd8	1272	50.8	3048	1	116.3	−139.62	−128.94
sctap2	953	35.0	872	0	33.3	8.50	4.86
sctap3	1221	59.5	1251	0	62.8	−2.46	−5.55
seba	224	1.4	222	0	1.2	0.89	14.29
shell	276	5.5	276	0	5.5	0.00	0.00
ship04l	236	4.8	236	0	4.8	0.00	0.00
ship04s	185	3.0	185	0	3.0	0.00	0.00
ship08l	487	20.9	487	0	20.9	0.00	0.00
ship08s	303	9.3	304	0	9.4	−0.33	−1.08
ship12l	919	51.1	913	0	51.1	0.65	0.00
ship12s	484	18.5	484	0	18.5	0.00	0.00
sierra	772	30.9	767	0	30.1	0.65	2.59
standata	86	1.3	86	0	1.3	0.00	0.00
standgub	86	1.4	86	0	1.4	0.00	0.00
standmps	271	4.4	265	0	4.4	2.21	0.00
tuff	270	6.1	270	0	6.1	0.00	0.00
woodw	1916	187.6	2040	0	210.1	−6.47	−11.99
80bau3b	8930	957.3	8223	0	923.6	7.92	3.52
Average:						−3.00	−0.23

8. Conclusions

We have presented several issues of an advanced and efficient (but still experimental) implementation of the revised simplex method for linear programming. The most important of its original features include:

- a new approximate steepest edge pricing method,
- efficient, numerically stable and reliable penalty method for finding an initial feasible solution,
- equal treatment of basic and non-basic variables (they are both allowed to take any feasible values) which facilitates avoiding degeneracy,
- ability to restart easily and reliably (as well as start from any combination of initial solution and basis[8]),
- ability to provide information useful in tracking down the source of infeasibility,
- a heuristic which reduces the infeasibility of the initial solution.

The results of numerical experiments presented in Section 7. have proven practicability, usefullness, robustness and reliability of our algorithms. The tests confirmed the theoretical speculations concerning the performance of some of our techniques. Some of the ideas presented here, specifically those regarding simplex method's restarts, were already put to the test of sequential solution of thousands and tens of thousands of middle and large scale linear problems arising in a certain stochastic program decomposition scheme[16]. We hope to be able to use this method successfully in many other novel and non-standard approaches to linear program solution (e.g. method of Wierzbicki [20]).

9. Acknowledgements

This work has benefited greatly from help and guidance of Prof. Andrzej Ruszczyński, who first introduced the idea of a simple approximation to steepest edge pricing. Thanks are due to Dr. Jacek Gondzio for sharing his invaluable expertise in the field of linear programming implemenation techniques. He also made many helpful comments during the preparation of this report. Last but not least, I would like to thank Prof. Andrzej Wierzbicki for encouragement and support of my research.

[8]It is possible to specify only an initial basis or only the solution — the algorithm will take care of finding the missing part of initial solution information.

References

[1] R. H. Bartels and Gene H. Golub. The Simplex method of linear programming using lu decomposition. *Communication of ACM*, 12:266–268, 1969.

[2] Robert E. Bixby. Implementing the Simplex method: the initial basis. *ORSA Journal on Computing*, 4(3):267–284, 1992.

[3] Robert E. Bixby. Progress in linear programming. *ORSA Journal on Computing*, 6(1):15–22, 1994.

[4] George B. Dantzig. *Linear Programming And Extensions.* Princeton, 1963.

[5] John J. Forrest and Donald E. Goldfarb. Steepest-edge Simplex algorithms for linear programming. *Mathematical Programming*, 57:341–374, 1992.

[6] John J. Forrest and J. A. Tomlin. Implementing the Simplex method for the optimization subroutine library. *IBM Systems Journal*, 31(2):11–25, 1992.

[7] David M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 1985.

[8] Donald E. Goldfarb and John K Reid. A practicable steepest-edge Simplex algorithm. *Mathematical Programming*, 12:361–371, 1977.

[9] Paula M. J. Harris. Pivoting selection methods of the DEVEX LP code. *Mathematical Programming Study*, 4:30–57, 1975.

[10] Bruce Murtagh. *Advanced Linear Programming, Computation and Practice.* McGraw–Hill, New York, 1981.

[11] J. L. Nazareth. *Computer Solution of Linear Programs.* Oxford University Press, 1987.

[12] John K. Reid. A sparsity-exploiting variant of the bartels-golub decomposition for linear programming bases. *Mathematical Programming*, 24:55–69, 1982.

[13] Andrzej Ruszczy@nski. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333, 1986.

[14] Andrzej Ruszczy@nski. Regularized decomposition of stochastic programs: algorithmic techniques and numerical results. Working Paper WP-93-21, International Institute for Applied Systems Analysis, 1993.

[15] Uwe H. Suhl and Leena M. Suhl. Computing sparse lu factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, 2:325–335, 1990.

[16] Artur @Swi@etanowski. Efficient solution techniques for two stage stochastic linear problems. Working paper, International Institute for Applied Systems Analysis. (in preparation).

[17] Artur @Swi@etanowski. A modern implementation of the revised Simplex method for large scale linear programming. Master's thesis, Institute of Automatic Control, Warsaw University of Technology, Warsaw, 1993. (in Polish).

[18] Artur @Swi@etanowski. SIMPLEX v. 2.17: an implementation of the simplex algorithm for large scale linear problems. user's guide. Working Paper WP-94-37, International Institute for Applied Systems Analysis, 1994.

[19] J. A. Tomlin. On scaling linear programming problems. *Mathematical Programming Study*, 4:146–166, 1975.

[20] Andrzej P. Wierzbicki. Augmented Simplex: a modified and parallel version of Simplex method based on multiple objective and subdifferential optimization approach. Working Paper WP-93-059, International Institute for Applied Systems Analysis, 1993.