Optimization Methods and Software, Vol. 17, pp. 717–745

Taylor & Francis
Taylor & Francis Group

LPAKO: A SIMPLEX-BASED LINEAR PROGRAMMING PROGRAM

SUNGMOOK LIM[a,*] and SOONDAL PARK[b]

[a]*Department of Information Exchange and Certification, Center for National Informatization, National Computerization Agency, NCA Bldg., 77, Mugyo-dong, Chung-ku, Seoul 100-170, Republic of Korea;*
[b]*Department of Industrial Engineering, Seoul National University, Shillim-dong, Kwanak-Gu, Seoul, 151-742, Republic of Korea*

(Received 19 June 2001; In final form 14 May 2002)

LPAKO is a public domain simplex-based linear programming program which can solve large-scale, sparse linear programming problems. It has been widely used in many applications and shows better performance than other public domain simplex-based programs. Several aspects considered in the development of LPAKO are described in this article such as the construction of initial basis, *LU* factorization of basis matrix, pricing rule, presolving/postsolving and other miscellaneous issues. At the end of the article, we introduce H. Mittelman's benchmark result which compares the performance of LPAKO with those of several simplex-based programs. We also compare LPAKO with CPLEX on the NETLIB test set.

Keywords: Linear programming; The simplex method; LPAKO

1 INTRODUCTION

Linear programming (LP) is one of the most useful and fundamental mathematical programming models which is used widely in operations research and in many other areas of science [25]. The usefulness of linear and mathematical programming models has been demonstrated over the last fifty years. Transportation planning, portfolio selection and productivity analysis are good examples of applications of

*Corresponding author. Tel.: +082-02-2131-0278. Fax: +082-02-2131-0299. E-mail: sungmook@nca.or.kr

ISSN 1055-6788 print; ISSN 1029-4937 online
DOI: 10.1080/1055678021000049381

mathematical programming in operations research [29]. In real life, many applications of mathematical programming models give rise to very large-scale LP problems. Therefore, it is very essential to have methods and their implementations which solve large LP problems efficiently.

The simplex method has been studied extensively since its invention in 1947 by G.B. Dantzig and still remains one of the most efficient methods for solving a great majority of practical problems. Although a number of variants of the simplex method have been developed, none of them has polynomial time complexity [38]. That is, the simplex method may require computational effort which grows exponentially with the size of the given problem. Hence, the simplex method is not considered efficient from a theoretical point of view contrary to its practical efficiency.

The first efficient algorithm of which computational requirement grows polynomially in problem size was invented by Khachian in 1979 and is called the ellipsoid algorithm. But the performance of this algorithm is poor for practical problems compared with the simplex method. In 1984, Karmarkar presented a new polynomial algorithm for LP which approaches the optimal solution from the interior of the feasible region [16]. Since this Karmarkar's seminal paper was published, many papers have been written about interior point methods [33, 40]. As their theoretical properties show, interior point methods work well for practical problems in real life and outperform the simplex method for very large-scale problems [22].

Despite its bad performance for large-scale problems, the simplex method often works better than interior point methods for middle or small sized problems. Also, for some classes of LP problems such as network-embedded problems, the simplex method requires much less computational effort than interior point methods. Furthermore, for the solution method of a series of related LP problems, the simplex method is more suitable than interior point methods because warm starting from an advanced solution is difficult for interior point methods whereas the simplex method has no difficulty in warm starting. One of the drawbacks of interior point methods is that they often produce a solution in optimal face other than a basic solution. In many applications of LP, a basic solution is more useful. The obvious remedy for this drawback is

combining interior point methods with a basis identification algorithm such as Megiddo's polynomial algorithm [24] and Bixby's crossover scheme [5]. For these reasons, the development of a simplex-based LP program is well justified.

Up to now, many LP packages have been developed for academic or commercial purposes, such as ILOG's CPLEX, IBM's OSL and SOL's MINOS. They often implement both the simplex method and one of the interior point methods. In this article, we deal with the development of LPAKO, a simplex-based LP program for which source code is available at the author's web site (*http://orlab.snu.ac.kr/software/lpako.html*). LPAKO has been widely used because it is in the public domain and shows superior performance compared to other simplex-based LP programs in public domain.

This article covers several important considerations in developing LPAKO. In Section 2, some notations are defined and the standard revised simplex method using LU factorization of basis matrix is stated. Presolving and postsolving procedures are covered in Section 3, and the initial basis construction procedure in LPAKO is explained in Section 4. The LU factorization of basis matrix and the update method of LU factors are described in Section 5. In Section 6, the pricing rule and anti-degeneracy technique adopted in LPAKO are presented. One artificial variable technique for constructing a phase 1 problem is introduced in Section 7. Finally, miscellaneous issues including scaling, tolerances, sensitivity analysis, hardware exploitation, callable library and user-interface are briefly covered in Section 8. The results of computational experiments comparing the performance of LPAKO with other simplex-based LP programs are presented in the end of this article.

2 ALGORITHM

In this section, we define some notations and then briefly outline the standard revised simplex method. We consider primal LP problems with bounded variables in the standard form:

$$\text{(LP)} \qquad \begin{aligned} \min \ & c^T x \\ & Ax = b \\ & l \le x \le u \end{aligned}$$

where $l, u, c, x \in R^n$, $b \in R^m$, and A is a sparse $m \times n$ real matrix with rank m. The vectors l and u may contain elements which are plus or minus infinity. Let B represent a starting feasible basis; B is a nonsingular $m \times m$ submatrix of A, g is the ordered set of size m consisting of indices of basic variables, i.e. g_i is the index of a column vector of A which is at position i in the basis. To B corresponds an m vector of basic variables x_B and an $n-m$ vector of nonbasic variables x_N. A basic solution is determined by fixing the nonbasic variables to one of its bounds (if they exist) and then solving the resultant linear system to determine x_B. If a nonbasic variable is free, i.e. $l_j = -\infty$ and $u_j = +\infty$, it is set to zero. Let a_i be the ith column of A and π, y be real m vectors. N represents a submatrix of A consisting of nonbasic columns.

Using the above notations and abbreviating many details, we state the standard revised simplex method using an LU factorization of basis matrix:

Step 0 (LU Factorization) Compute $LU = PBQ^T$, where L, U are lower, respectively upper triangular matrices and P, Q are row, respectively, column permutation matrices.

Step 1 (BTRAN: Simplex multiplier calculation) Compute the shadow price vector π: solve $B^T\pi = c_B$, where c_B is the basic components of c.

Step 2 (Optimality test) If the solution is dual feasible, an optimal solution is found, so terminate. Dual feasibility can be tested by computing the reduced costs, $\bar{c}_j = c_j - \pi^T a_j$: If for all nonbasic variables x_j, $\bar{c}_j \geq 0$ if variable x_j is at lower bound or $\bar{c}_j \leq 0$ if variable x_j is at upper bound, stop.

Step 3 (Price) If the test in Step 2 failed, select a nonbasic variable x_q not dual feasible to enter the basis according to a pivot column selection rule.

Step 4 (FTRAN) Update the entering column: solve $By = a_q$ if $\bar{c}_q < 0$ or $By = -a_q$ if $\bar{c}_q > 0$ by using the LU factorization and retain the spike $L^{-1}a_q$.

Step 5 (CHUZR) Determine a blocking variable to leave the basis in position p and a step length Θ according to a pivot row selection rule or switch x_q to the other bound. If no such variable exists, the LP is unbounded, so stop.

Step 6 (LU Factor update) If the basis changed, replace column p of U by the spike $L^{-1}a_q$. Let r be the position of the last nonzero in the permuted spike. Move columns $p+1$ to r one position to the left and column p to position r. This creates the matrix $H=[U_1, L^{-1}a_q, U_2]$. Restore upper triangularity of H. If there are numerical problems or too many elements in the new L or U, go to Step 0, otherwise update g, x_B and x_N to reflect the basis change and go to Step 1.

In LPAKO, the dual simplex method is also implemented. We omit the description of the dual simplex method and refer the reader to [7]. The implementation issues concerning the dual simplex method are almost the same as with the primal simplex method, so we state our results in terms of the primal simplex method.

3 PRESOLVING AND POSTSOLVING

It is often that a large-scale LP problem may contain many constraints which are redundant or cause infeasibility on account of inefficient formulation or some errors in data input. Presolving or preprocessing is a series of operations which removes the underlying redundancy or detects infeasibility in the given LP problem. It is essential for the speedup of an LP system solving large-scale problems to implement presolving techniques. Various presolving techniques for the simplex method and interior point methods were suggested by many authors [1, 13, 22]. In the papers, the great effects of presolving for the simplex method and interior point methods are well appreciated by the results of computational experiments.

In LPAKO a large number of presolving techniques are implemented [21]. For the detection of infeasibility and unboundedness, the following methods are implemented:

- Detection of infeasibility in empty rows.
- Detection of infeasibility in singleton rows.
- Detection of infeasibility by examining the value-ranges of rows.
- Detection of unboundedness in empty columns.
- Detection of unboundedness by examining the value-ranges of columns.

Techniques for problem size reduction implemented in LPAKO can be categorized as follows:

- Basic process.
- Substitution.
- Removal of inherent redundancy.

The basic process contains

- Elimination of empty rows and columns.
- Elimination of singleton rows and free rows.
- Processing singleton columns and fixed columns.

The substitutions occur in

- Doubleton rows.
- Sparse rows containing (implied) free variables.

Inherent redundancy is removed by

- Fixing variables using information of reduced costs.
- Elimination of redundant rows.
- Tightening bounds on variables.

A state of the art presolver must have the option of eliminating balance constraints such as $x_j = \sum x_i$. This feature is missing in the current version of LPAKO (ver 4.8f), but the authors have the plan to implement this feature in the next version of LPAKO. A preliminary experiment shows that the balance constraints reduction technique improves greatly the performance of the presolver in LPAKO.

For the recovery of an optimal solution for the original problem from an optimal solution for the preprocessed problem, a postsolving procedure must be implemented. In LPAKO, all presolving processes are logged in a data structure, and after the end of the solution method an optimal solution for the original problem is obtained by a postsolving procedure using the log.

4 CONSTRUCTION OF INITIAL BASIS

Not so many research papers have been published on constructing an advanced initial basis in contrast with other performance enhancing

techniques for the simplex method. But, the advantages of using advanced initial basis have been well appreciated and most of high performance simplex-based LP systems implemented their own strategies for creating advanced initial bases.

Bixby [4] introduced a penalty function used to evaluate columns for joining initial basis. He constructed the function considering the distance between bounds and objective coefficients of the columns of the constraint matrix. By computational experiments on NETLIB [10] sets, Bixby showed that his method is very effective. On the other hand, Maros [23] extended the traditional crashing technique suggested by Carstens [28] and showed that his method is superior to Bixby's. In the early development of LPAKO, Seo and Park [34] suggested an initial basis construction method in which logical variables are first included in initial basis for their merits of sparsity and numerical stability, and a symbolic method is used for the inclusion of structural columns in favor of sparser columns.

Two desirable properties of the initial basis are emphasized in the current version of LPAKO [20]. They are the sparsity and the optimality of initial basis. The sparsity of initial basis determines the initial performance of the simplex method, and the optimality of initial basis may help reduce the number of iterations. The sketch of the procedure by which LPAKO constructs initial basis is as follows: slack or surplus columns are considered first for their good sparsity and numerical stability. A certain portion of the slack or surplus columns are included in initial basis in non-increasing order of a certain criterion measuring optimality. After that, structural columns are included as many as possible maintaining the nonsingularity of basis by so-called 'non-overlapping nonzeros' method. The effort for inclusion of structural columns is in the ground that less inclusion of artificial columns means fewer iterations of the simplex method.

Optimality of Columns

In LPAKO, the optimality of a column is determined by the criterion implied by Sherali *et al.* [35]. Let us consider the following

LP in canonical form:

$$\text{(CLP)} \qquad \begin{aligned} &\min\ h^T w \\ &s.t.\ q \le Dw \le p \\ &\quad l^w \le w \le u^w, \end{aligned}$$

where l^w, u^w, h, $w \in R^k$, p, $q \in R^m$, and D is a $m \times k$ real matrix. The vectors l^w, u^w, p and q may contain elements which are plus or minus infinity. This form will be later converted equivalently to (LP) in standard form by adding some slack and surplus variables where $A=[D, T]$, $c=(h, 0)^T$, $x=(w,s)$, T is a $m \times (n-k)$ matrix consisting of appropriate slack and surplus columns, and s is a $(n-k)$-vector of slack and surplus variables.

In the above problem, the feasible region consists of m hyperplanes, $Dw \ge q$, m hyperplanes, $-Dw \ge -p$, k hyperplanes, $w \ge l^w$, and k hyperplanes, $-w \ge -u^w$. By the KKT optimality condition, the objective vector h is in the positive cone of the normal vectors of the hyperplanes which are active at an optimal vertex. By this condition, we can expect that the normal vectors of the hyperplanes which are active at an optimal vertex make acute angles with the objective vector. The angle, θ_i, which ith constraint makes with the objective vector can be calculated as follows:

$$\theta_i^1 = \cos^{-1}\left(\frac{D_i h}{\|D_i\|\ \|h\|}\right), \quad \theta_i^2 = \cos^{-1}\left(\frac{-D_i h}{\|D_i\|\ \|h\|}\right), \quad \theta_i = \min\{\theta_i^1, \theta_i^2\},$$

where D_i is the ith row of D and $\theta_i^1(\theta_i^2)$ is set to plus infinity if $q_i(p_i)$ is plus or minus infinity.

As θ_i becomes smaller, the possibility that the constraint $q_i \le D_i w \le p_i$ is active at an optimal vertex is larger, that is the possibility that the surplus or slack variable s_i is nonbasic at an optimal vertex is larger. In LPAKO, all slack and surplus columns are considered in the order created by the above optimality criterion.

Detecting Linearly Independency – 'Non-overlapping Nonzeros' Method

Partial application of Gaussian elimination can detect the linearly independency, but it requires much computational effort. Therefore,

the symbolic method, so-called 'non-overlapping nonzeros' method [34], is used in LPAKO. The main idea in the method is that a structural column, whose nonzero-pattern does not overlap with the nonzero-patterns of previously selected basic columns, can be included in basis. For this method, two one-dimensional arrays should be used for tracking the accumulated nonzero-pattern of constructed basis matrix and storing pivot positions while inverting the basis matrix. This method can reduce the computational effort greatly and can detect larger number of linearly independent columns than the expected.

Overall Procedure

The overall procedure by which an initial basis is constructed in LPAKO, is as follows:

Step 1 Let g be the basis list initially set to null. Set all components of two arrays *nnz* and *pivot* of size m to zero.

Step 2 Compute θ_i for all constraints except equality constraints where $p_i = q_i$.

Step 3 Put slack or surplus columns into an ordered set L in the increasing order of the corresponding θ_i.

Step 4 For an arbitrary real number $0 < \alpha < 1$, extract $[\alpha|L|]$ slack or surplus columns from the top of the set L, and put them into g. For all i, set both *nnz*(i) and *pivot*(i) to one if at least one of columns in g has nonzero at position i, where *nnz*(i) and *pivot*(i) mean ith components of the arrays *nnz* and *pivot*, respectively.

Step 5 Make the set L empty, and put the structural columns into the set L in the increasing order of the corresponding number of nonzeros. Set k to 1.

Step 6 If the basis list g is complete, then stop.

Step 7 If the nonzero-pattern of $L(k)$ does not overlap with that of *nnz*, then put $L(k)$ into g, where $L(k)$ is the kth component of the set L. Let j be the index of the element of $L(k)$ which has the largest absolute value. Set *pivot*(j) to 1, and for all i, set *nnz*(i) to 1 if $L(k)$, is nonzero.

Step 8 If k is less than n, then set $k = k + 1$, and go to the Step 6.

Step 9 If the basis list is not complete, add artificial columns to g to complete basis as follows: for all i, add an ith unit vector to g if $pivot(i)$ is zero.

In LPAKO, the value of α is set to 0.5.

5 *LU* FACTORIZATION OF BASIS MATRIX

In every iteration of the simplex method, a linear system of the form $Bx = p$ or $B^T y = q$, must be solved. The computational effort of solving the system occupies over 70–80% of overall computations of the simplex method [36]. Thus an efficient implementation of the solution procedure can enhance the performance of the simplex method program greatly. In the most efficient simplex implementations, LU factorization of basis matrix is used for the reasons of its sparsity-exploiting ability, less computational requirement than explicit inverse calculation, and numerical stability. In this section, the LU factorization algorithm and the update algorithm of LU factors in LPAKO are explained.

In LPAKO, Gaussian elimination is used for LU factorization in which Markowitz ordering is used for increasing sparsity of LU factors [18]. For the numerical stability of LU factors, dynamic threshold pivoting technique [6] is implemented. For fast factorization, the symbolic factorization, that is triangularization, is performed before the actual numerical factorization.

The efficiency of LU update algorithm depends on the data structure of L and U, and reinversion timing. The modified Forrest–Tomlin method suggested by Suhl and Suhl [37] is implemented in LPAKO, and Gustavson's data structure [6] for U is implemented. The idea of Suhl is adopted for the storage of L, by which fast FTRAN is possible.

Reinversion (Refactorization) should be performed for the numerical stability and error-adjustment of LU factors. The timing of reinversion takes great effect on the speed and stability of the simplex method program. In LPAKO, the spasity and the growth factor of U determines reinversion timing.

5.1 *LU* Factorization

The storage form of basis inverse has evolved from the explicit form over the product form to the *LU* Factorization (LUF) computed by Gaussian elimination. The product form of inverse represents the inverse as an ordered set of elementary matrices whereas the LUF stores the factors of the basis matrix B produced by *LU* factorization. It is well known that the LUF is generally sparser than the product form of inverse. Often a large part of the basis matrix can be triangularized by permuting it. In LPAKO, we implemented triangularization algorithm following the principles presented by Orchard-Hays. After triangularization, the matrix is permuted to the form shown in Fig. 1, where N is the nucleus which will be numerically factorized.

To maintain numerical stability, it is very important that the chosen pivot element is not too small with respect to the size of other elements in the active submatrix. There are various strategies for pivot selection such as full pivoting and partial pivoting. Full pivoting guarantees numerical stability. In practice, however, partial pivoting provides enough numerical stability on nearly all problems at a much lower cost than full pivoting. For sparse matrices, both approaches are not appropriate since they do not restrict fill-ins and may make the factors very dense as a result [36]. Threshold pivoting allows more freedom to select a pivot based on sparsity grounds without too much consideration on numerical stability: let I^s be the set of row indices which were not pivotal until stage s, J^s be the set of column indices which were not pivotal until stage s, B^s be the active submatrix at stage s, i.e., that part of the matrix which has not yet been factorized. The elements of B^s are denoted by $b^s_{i,j}$. In a given row i of the active submatrix at stage s an element $b^s_{i,j}$ is a candidate for pivoting, if and

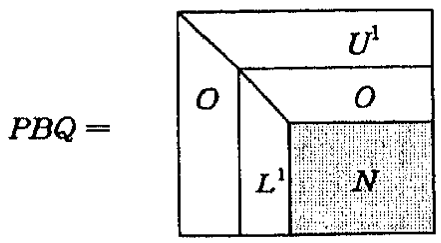

FIGURE 1 Triangularization of basis matrix.

only if it satisfies the following stability criterion:

$$|b^s_{l,j}| \geq u \times \max\{|b^s_{i,k}| : k \in J^s\} \qquad \text{(S)}$$

where u is a real parameter in the range $0 < u \leq 1$. It was recommended in [6] that a value of u around 0.1 is a good compromise between numerical stability and sparsity. In LPAKO, however, we used 0.01 or 0.02 for the value of u, and obtained good performance for large-scale LP problems with sparse matrices. Among the elements which satisfy the threshold pivoting criterion we want to choose as pivot one with the smallest Markowitz count in order to maintain sparsity of LU factors. The stability of the factorization is controlled by the threshold pivoting tolerance u and the drop tolerance *tol_zero*. A computed value in L or U is set to zero if its absolute value is smaller than the tolerance *tol_zero*. In LPAKO, the default values are $u = 0.02$ and $tol\_zero = 10^{-14}$. During the computations we monitor the absolute largest element d in L and U. If the ratio of d to the largest absolute element of B exceeds a given value the factorization is cancelled and the value of u is multiplied by 1.2 until it reaches 1.0.

5.2 *LU* Update

Markowitz used the LUF in reinversion, but the product form of inverse during simplex iterations. Bartels and Golub [2] found a technique of updating the LUF after each step of the simplex method. Forrest and Tomlin [9] found an efficient update technique and showed experimentally for large-scale LP problems a smaller growth rate in the number of nonzeros compared to the product form of inverse update.

In most commercial packages or academic programs in public domain, the modified Forrest–Tomlin method is implemented for the update of LU factors. It is well known that the method enables a fast implementation of LU update and is not so bad for numerical stability, compared with Reid's method [32] or Bartels–Golub's method.

The modified Forrest–Tomlin method exploits the sparsity of spike columns. Differently from the original Forrest–Tomlin method, it permutes the upper triangular matrix so that the spike column is

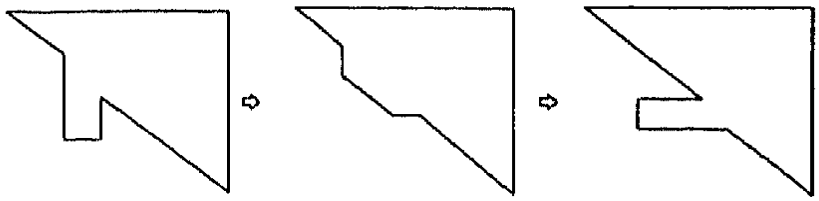

FIGURE 2 Modified Forrest–Tomlin LU update.

placed on the column corresponding to the last nonzero element in the spike column. The situation is depicted in Fig. 2.

For the fast implementation of the modified Forrest–Tomlin LU update, two techniques suggested by Suhl and Suhl [37] are adopted in LPAKO. First, the nonzero pattern of spike column is stored in stack data structure so that the linear search over the spike column is avoided. Second, the column index of the eliminated row is stored in stack data structure so that the scatter/gather technique can be applied during Gaussian elimination.

6 PRICING RULE AND ANTI-DEGENERACY TECHNIQUE

6.1 Pricing Rule

The most important improvements in many primal simplex implementations over the last several years have come in the pricing step [3]. It is well known that much of the credit for these improvements is due to steepest-edge type simplex methods developed by Harris [14] and Goldfarb and Reid [12]. In our computational experiments, both methods outperform greatly the traditional Dantzig's rule in most problems. In LPAKO, a hybrid pricing which combines Dantzig's rule and the dynamic steepest-edge pricing is implemented. It is described first how Dantzig's rule and the dynamic steepest-edge pricing are implemented in LPAKO, and next the hybrid scheme is described.

Dantzig's Rule

In the standard simplex method, the entering variable x_j is selected as the one that has the least reduced cost, which is called the most-negative rule or Dantzig's rule. Geometrically reduced cost measures

the change in the objective per unit change of the current solution along the corresponding coordinate axis. In almost all traditional LP systems, multiple-partial pricing technique, of which objective is to reduce time for pricing, is embedded in Dantzig's rule, which is also true in LPAKO. In the implementation of multiple-partial pricing, the adaptive choice of the size of candidates list and the suitable partitioning of variable set affect the performance greatly. In LPAKO, the objective improvement and the degree of degeneracy are monitored in every iteration, and the parameters in multiple-partial pricing are dynamically adjusted according to the monitoring results: if the rate of the objective improvement is continuously below 5% for 100 iterations, or the degree of degeneracy (which is computed as the portion of basic variables at their bounds) is continuously above 80% for 100 iterations, the size of candidates list (S_P) and the size of multiple choice of entering columns (S_M) are doubled. The initial values of S_P and S_M are $(100 \times n)/nze(A)$ and 5, respectively, and the upper bounds of those are n and 10, respectively, where $nze(A)$ is the number of nonzeros in A.

Dynamic Steepest-Edge Pricing

The steepest-edge pricing measures the improvement of the objective in the space of all variables whereas Dantzig's rule only measures in the space of nonbasic variables. We call "the reference framework" of a given pricing rule as the variable space in which the objective improvement is measured. That is, the reference framework in the steepest-edge pricing is the set of all variables whereas that in Dantzig's rule is the set of nonbasic variables.

The exact steepest-edge pricing involves computation of the norm of each updated nonbasic column which makes the method impractical. To make the steepest-edge pricing attractive, Forrest and Goldfarb [8] suggested various approximate implementations of the method including the dynamic steepest-edge pricing. Devex pricing [14], developed by Harris, can also be seen as a method which implements the steepest-edge pricing approximately.

The dynamic steepest-edge pricing implemented in LPAKO enlarges the reference framework gradually as iteration goes. Initially the reference framework is set to the set of nonbasic variables as in

Dantzig's rule, and in every end of iteration the leaving variable is added to the reference framework. After enough iterations, the dynamic steepest-edge pricing is almost the same as the exact steepest-edge pricing. A large amount of computational burden of the dynamic steepest-edge pricing can be compensated by the use of interim computational results, such as $B^{-T}e_i$ where e_i is the ith unit vector, in the update of reduced costs and dual solution. The periodic re-initialization of the reference framework (every 1000th iteration in the case of LPAKO) can also be helpful for lessening the computational burden. Full pricing can also be adopted without not so much increase of computational effort because the update of reduced costs can be easily done in the dynamic steepest-edge pricing [30].

Hybrid Pricing

The default primal pricing rule in LPAKO is a hybrid method which combines a multiple-partial pricing with Dantzig's rule and a full pricing with the dynamic steepest-edge pricing. The code begins with the cheap multiple-partial pricing, and based on the progress in the objective and the relative cost of the linear algebra per iteration, conditionally switches to the dynamic steepest-edge pricing. The switching criteria involve the computation of the following values:

$$\Delta_{\text{rel}} z = \frac{|z_{\text{prev}} - z_{\text{cur}}|}{z_{\text{prev}}}, \qquad R_p = \frac{t_{\text{st}}}{t_{\text{red}}},$$

where z_{cur}, z_{prev} are the objective values at the current iteration and the previous iteration, respectively, and t_{st}, t_{red} are the average computation times per iteration using Dantzig's rule and the dynamic steepest-edge rule, respectively. These values, $\Delta_{\text{rel}} z$ and R_p measure the relative improvement of the objective value after one iteration and the ratio of the computational burden of the dynamic steepest-edge rule to that of Dantzig's rule. LPAKO switches to the dynamic steepest-edge pricing when $\Delta_{\text{rel}} z$ is continuously below 0.01 for 100 iterations, but switches back to Dantzig's rule when R_p is over 7.0.

This scheme is based on the following observation: generally, most of the basic variables are slack or surplus in the initial phase of the simplex method, so the basis has few elements and the representation

of the inverse can be computed easily. Therefore, most of the time in an iteration will be spent selecting an incoming variable. Additionally, a multiple-partial pricing with Dantzig's rule works not so much worse than a full pricing with the dynamic steepest-edge pricing in initial iterations. Hybrid approaches like this have been adopted in several simplex-based LP systems [3].

6.2 Anti-degeneracy Technique

The anti-degeneracy technique implemented in LPAKO is a variant of the EXPAND procedure which was originally invented by Gill *et al.* [11]. In conventional LP terminology, the EXPAND procedure is a pivot row selection method that specifies the choice of pivot row in the simplex method. The "maximum pivot" property of Harris row-selection method [14] is retained, and permitting infeasibility in nonbasic variables removes the difficulty with traditional implementations of the Harris procedure. The EXPAND procedure is summarized by the following pseudo-code:

Given the current basic solution x, an edge direction p, and predetermined tolerances δ, τ.
Procedure EXPAND $(x, p, l, u, \delta, \tau)$
$(\alpha 1, r1) \leftarrow radio\_test(x, p, l - \delta e, u + \delta e)$; (first pass)
$r \leftarrow 0$; $p_{\max} \leftarrow 0$;
for $j = 1$ until n do (second pass)
  $\alpha_j \leftarrow step\ (x_j, p_j, l_j, u_j)$;
  if $\alpha_j \leq \alpha 1$ and $|p_j| > p_{\max}$ then
  $\gamma \leftarrow j$; $\alpha 2 \leftarrow \alpha_j$; $p_{\max} \leftarrow |p_j|$;
  end if
end for
$\alpha_{\min} \leftarrow \tau / |p_r|$; (minimum acceptable step)
$\alpha \leftarrow \max\{\alpha 2, \alpha_{\min}\}$
end of EXPAND

In LPAKO, the EXPAND procedure is not applied in every iteration. Only when the degree of degeneracy is seen to be excessive, the EXPAND procedure is involved. In our experience, infeasibility problem is considerable when the EXPAND is invoked in every

iteration. Contrary to the original EXPAND procedure, the values of nonbasic variables are not stored explicitly in LPAKO [30].

7 ONE ARTIFICIAL VARIABLE TECHNIQUE

In the traditional primal simplex method, a two phase method is mainly used to find an initial starting basis which is primal feasible. Various methods have been suggested for constructing the phase 1 problem such as the composite simplex method developed by Wolfe [39]. In the composite simplex method, the sum of primal infeasibility of all variables are measured at each step, and based on the infeasibility phase 1 objective function is constructed. The phase 1 problem is constructed as follows:

$$\begin{array}{ll} \min \tilde{c}^T x & \\ s.t.\ \ Ax = b, & \tilde{c}_i = \begin{cases} -1, & \text{if } x_i < l_i \\ 1, & \text{if } x_i > u_i \\ 0, & \text{otherwise} \end{cases} \\ \quad l \leq x \leq u & \end{array}$$

Wolfe published two methods for finding a feasible starting basis: one is the composite simplex method and the other is the extended composite simplex method, also known as "minimizing the sum of the infeasibilities". The two methods differ from each other in the pivot row selection. The extended composite simplex method was reinvented from time to time and has the potential to allow non-degenerate pivot steps in phase 1, even when some variables in the basis are at their bounds. It is used (at least as an option) in all major simplex codes including LPAKO. In LPAKO, however, another technique proposed in the next paragraph is also implemented and used as default [19].

In the proposed method, an arbitrary basis is selected and the corresponding values of basic variables are set to satisfy the bound restrictions. The values of nonbasic variables are set to their bounds arbitrarily. Then, the basis does not satisfy the linear constraints, $Ax = b$, but the corresponding basic variables satisfy the bound restrictions. To adjust the infeasibility arising in the linear constraints, an artificial column is added to the problem. If arbitrary vectors $\bar{x}_B$, $\bar{x}_N$ satisfy the bound restrictions, the one artificial column can be added to

the (LP) as follows:

$$\text{(LP1)} \qquad \begin{array}{ll} \min & x_A \\ s.t. & Bx_B + Nx_N + (b - B\bar{x}_B - N\bar{x}_N)x_A = b \\ & l \leq x \leq u, \quad 0 \leq x_A \leq 1 \end{array}$$

In the above (LP1), we may take a starting solution as $x_B = \bar{x}_B$, $x_N = \bar{x}_N$, $x_A = 1$.

In the (extended) composite simplex method, the objective coefficients are changed at each iteration according to the infeasibility status, so that the update procedure of reduced costs is not easy. In the other hand, if the one artificial variable technique is used, the objective coefficients are fixed, so that the update of shadow prices and reduced costs can easily be performed at each iteration. Normalized pricing such as the steepest-edge pricing requires the update of reduced costs to reduce the computational burden. So, the one artificial variable technique is more efficient than the (extended) composite simplex method when used with normalized pricing strategies. Also, our computational experiments show that the one artificial variable technique requires less simplex steps than the (extended) composite simplex method does.

8 MISCELLANEOUS ISSUES

8.1 Scaling and Tolerances

Orchard-Hays said that the purpose for using scaling is to adjust the values regularly so that the given problem becomes numerically more stable [28]. If there exist quite small or large values compared to other values in the input data of a problem, numerical instability may occur. Therefore, it is necessary to adjust the values regularly in order to make the problem numerically more stable. This procedure is called scaling. Three scaling methods are implemented in LPAKO [27]. These are:

- *arithmetic mean method*: calculates the arithmetic mean of absolute values for each row and column, and divides the elements in the corresponding row and column by those values, respectively.

- *equilibration method*: finds the maximum of absolute values for each row and column, and divides the elements in the corresponding row and column by those values, respectively. In this method, all elements of the transformed matrix have values between -1 and 1.
- *geometric mean method*: calculates the geometric mean of minimum and maximum absolute values for each row and column, and divides the elements in the corresponding row and column by those values, respectively.

The default scaling method of LPAKO is to apply the geometric mean method followed by the equilibration method. Some optimal scaling methods have also been implemented but they showed worse performance than the above heuristics, so they are excluded in LPAKO.

To adjust numerical errors, several tolerances are used. The first is *tol_feas*, which is used to decide primal or dual feasibility; the current basis is primal feasible if its corresponding primal basic solution satisfies $l - tol\_feas \leq x \leq u + tol\_feas$, and dual feasible if its corresponding dual solution satisfies $c_j - A_j^T \pi \geq -tol\_feas$ for each nonbasic variable x_j at lower bound and $c_j - A_j^T \pi \leq tol\_feas$ for each nonbasic variable x_j at upper bound. The second tolerance is *tol_zero* which is used to discriminate zero from nonzero. That is, if a value in A or in the LU factors of basis is smaller than *tol_zero*, then it is considered as zero and excluded from computations. The third tolerance is *tol_stab* used in pivot row selection. Pivot element whose absolute value is smaller than *tol_stab* is never accepted in LPAKO. Other miscellaneous tolerances are also used to maintain stability of LPAKO.

In LPAKO, the tolerances mentioned in the above paragraph are set as:

- $tol\_feas = 10^{-8}$
- $tol\_zero = 10^{-14}$
- $tol\_stab = 10^{-5}$.

8.2 Sensitivity Analysis

Sensitivity analysis is the study of the behavior of the optimal solution with respect to changes in the input parameters of the original

optimization problem. The information of sensitivity analysis can be of tremendous importance in practice, where parameter values may be estimates [29]. Many LP packages can perform sensitivity analysis, but they produce some different results according to their sensitivity analysis algorithms [15]. To overcome this confusion, LPAKO offers several sensitivity analysis algorithms and the user can select any of them Three sensitivity analysis algorithms are implemented on LPAKO. They are the basic sensitivity analysis (BSA) [29], the positive sensitivity analysis (PSA) [41] and the optimal partition sensitivity analysis (OSA) [15]. BSA is the traditional method using optimal bases which finds the characteristic regions of parameters that do not change the optimal basis. In each LP problem a strictly complementary solution exists. Such a solution uniquely determines the optimal partition of the LP problem. OSA is an approach to sensitivity analysis using optimal partitions. PSA is a more general approach than OSA, which finds the characteristic regions of parameters that maintain nonzero patterns of optimal solutions. To compute either PSA or OSA, all optimal basic solutions are needed. For PSA and OSA, all alternative optimal solutions are generated using the revolving-door algorithm [26]. In LPAKO, BSA, PSA and OSA on objective coefficients and right-hand side can be performed, and BSA on the matrix coefficients can also be performed.

8.3 Exploitation of Computer Hardware

Modern advances of computer hardware have made great improvement of optimization softwares possible. Along with the advances of hardware, software programming skills are improved greatly. Most modern commercial LP packages are written with various programming techniques. These programming techniques affect the performance of LP packages greatly along with other algorithmic elements.

In LPAKO, various hardware-exploiting techniques are applied such as loop unrolling, direct addressing, scatter/gathering, etc. [31]. Such techniques are automatically applied at compile time by using optimization option of compiler, but our computational experiment shows that the explicit use of those techniques at source level help enhance the performance of the program.

8.4 Callable Library

To be used for the subsystems of other optimization systems, an LP system should offer reliable and flexible callable libraries. Like every reasonable LP-system, LPAKO offers callable libraries of which structure is as follows:

- Data input–output and problem definition routines
- Optimization routines
- Problem modification routines
- Query processing routines
- Parameter setting routines

The interface between LPAKO callable library and user-written program can be depicted as in Fig. 3.

Using the callable library of LPAKO, users can write their own optimization programs which need LP solution system. Users can also interact directly with LPAKO computation core routines through the "user interface" component depicted in the Fig. 3. The user interface in LPAKO is described in the next subsection.

8.5 Command-line Based User Interface

It is essential for LP solution systems to have well designed user-interface so that it may be used easily and correctly by common users. In LPAKO, a command-line based user-interface is implemented [17].

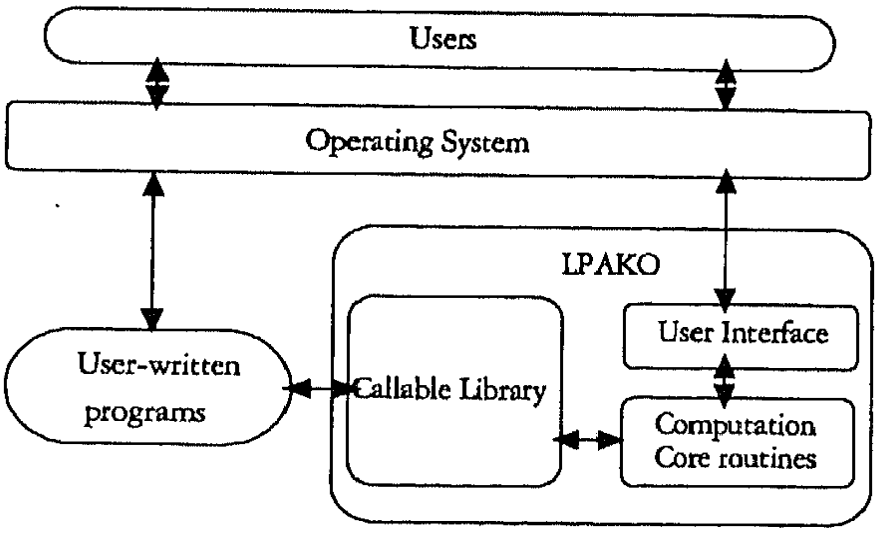

FIGURE 3 The structure of LPAKO.

TABLE I Categorization of required functionalities

Functionalities	*Contents*
Help	Help users to understand various commands
Data input	Input user-written problem data
Optimization	Solves the given problem and analyzes the optimal solution
Problem information	Shows problem or solution information
Parameter setting	Sets various parameters
Miscellaneous functions	Inputs starting point information
	Introduces additional constraints or variables
	Executes OS commands
Termination	Terminates the program

TABLE II Commands in the user interface of LPAKO

Commands	*Descriptions*
help	Shows the information about various commands.
set	Sets various parameters.
read	Reads user-written data files.
enter	Read user-input through keyboard type.
wstart	Reads starting point informations.
display	Prints input–output information and additional information.
optimize	Optimizes a given problem.
change	Modifies the given problem.
xecute	Executes OS commands.
quit	Quits the program.

We categorized various functionalities that the general purpose LP package should offer in Table I.

Based on the categorization of functionalities in Table I, various commands are implemented in LPAKO as in Table II. Various subcommands subordinate to the main commands are also implemented in LPAKO, but their description is omitted in this article.

8.6 Data Input

In LPAKO, the MPS data format, which is the international standard LP data format, is supported. Additionally, LPAKO has its own columnwise data format. The MPS reading routines in LPAKO can read huge MPS data files in reasonable time. The essential technique is *Dynamic Hashing* which makes the complexity of the string matching of row and column name $O(1)$. The MPS reading time of LPAKO compares well with CPLEX or other commercial systems.

9 COMPUTATIONAL RESULTS

There are many simplex-based LP solution systems in public domain, which have been developed for academic purpose. In this section, we present the result of a computational test which benchmarks various simplex-based LP programs including LPAKO.

In his website *http://plato.la.asu.edu/bench.html*, H. Mittelmann has performed benchmark of some simplex-based LP solvers since several years ago. In his benchmark, several public domain simplex-based LP solvers are tested and compared with each other and with two codes which are not in public domain. Public domain solvers are LP_SOLVE 3.0, SOPLEX 1.0, SIMPO, and LPAKO 4.8f. MOMIP 2.3, SNOPT 5.3 and MINOS 5.5, which are not in public domain, are compared with them. We summarize the contents of H. Mittelmann's benchmark web pages.

All codes were run on the same machine (HP735-99, 192 MB memory). All were run in default mode except LP_SOLVE which was run with the "-s" option. LP_SOLVE and MOMIP can solve mixed integer optimization problems. SOPLEX was run in the 2 versions C/R, column respectively row-wise orientation. Extensive results for the individual codes are usually included with the distribution. In Table III is a comparison for a sample of medium-sized problems. Times given are user times in seconds. An "*" means some failure such as poor accuracy; &: "infinite lower bounds not implemented"; #: "quadruple precision version used".

The MPS-datafiles for all testcases are in one of (source given in Table III, first column)

(1) ftp://ftp.netlib.org/lp/data/ (and ./kennington)
(2) ftp://ftp.netlib.org/lp/infeas/
(3) http://www.caam.rice.edu/~bixby/miplib/rniplib.html
(4) ftp://ftp.sztaki.hu/pub/oplab/LPTESTSET/MISC/ (and ../PROBLEMATIC).

As can be seen in the benchmark, it can be said that LPAKO outperforms all other codes except SOPLEX and it shows similar performance with SOPLEX.

Next, we compare the performance of LPAKO with a commercial system, CPLEX ver 6.5.2. The two programs were run on the same

TABLE III Computational comparison among various simplex-based LP solvers

S	*Problem*	*LP_SOLVE*	*LPAKO*	*MOMIP*	*MINOS*	*SNOPT*	*SOPLEX_C/R*	*SIMPO*
1	80bau3b	339	100	151	94	164	48/124	319
1	bnl2	*#	13	184	55	80	15/25	155
1	ceria3d	180	1	21	42	48	3/13	&
1	d2q06c	*	121	998	658	1158	417/2840	8417#
1	d6cube	132	113	685	508	657	212/57	103
3	dano3mip_lp	*	3187	*	939	658	3722/11745	140343#
1	degen3	840 (-degen)	30	285	85	131	37/40	226
4	delf024	*	56	128	59	61	27/104	&
1	df1001	*	2044	*	83724	*	2809/5440	14956
3	dsbnip_lp	*	12	16	12	*	7/23	&
1	fit2d	1148	315	71	145	161	1669/1111	2510
1	fit2p	4805	288	5261	611	878	116/1007	25581
4	gen4	*	*		16390	21177	1271/3090	*#
1	greenbea	*	78	658	359	504	311/271	816
1	greenbeb	*	54	481	202	*	183/256	&
1	ken-11	2611	1291	3712	1378	*	257/363	5639
2	klein2	*	1	2	3	3	1/1	7
4	l30	*	*	*	63397	84558	1899/40914	&
4	large002	*	107	308	87	142	46/151	&
1	maros-r7	*	513	353	142	253	678/494	*#
1	nl	*	1341	6326	2328	3115	334/1185	*#
4	nsir2	417	111	843	131	159*	36/6	105
1	osa-07	95	24	35	48	28	46/25	55
1	pds-06	1032	221	2055	1733	446*	118/1002	570
1	pilot. ja	*	22	*	35	59	159/89	&
1	pilot87	*	450	9262	1578	2312	1697/4959	*#
3	qiu_lp	40	2	25	6	8	6/1	7
4	route	*	994	*	7066	8373	1631/2408	78
3	seymour-lp	*	170	*	829	967	86/867	571
1	stocfor3	14927	434	*	1372	*	449/346	1415

TABLE IV The characteristics of the test problems

Problem	*(1)*	*(2)*	*(3)*	*Problem*	*(1)*	*(2)*	*(3)*
bnl2	2324	3489	13999	maros-r7	3136	9408	144848
25fv47	821	1571	10400	nesm	662	2923	13298
80bau3b	2262	9799	21002	pilot	1441	3652	43167
cycle	1903	2857	20720	pilot87	2030	4883	73152
czprob	929	3523	10669	pilot. ja	940	1998	14698
d2q063c	2171	5167	32417	pilotnov	975	2172	13057
d6cube	415	6184	37704	scsd8	397	2750	8584
degen3	1503	1818	24646	sctap3	1480	2480	8874
fitld	24	1026	13404	ship081	778	4283	12802
fitlp	627	1677	9868	ship12l	1151	5427	16170
fit2d	25	10500	129018	ship12s	1151	2763	8178
fit2p	3000	13575	50284	stocfor3	16675	15695	64875
greenbea	2392	5405	30877	truss	1000	8806	27836
geenbeb	2392	5405	30877	woodlp	244	2594	70215
maros	846	1443	9614	woodw	1098	8405	37474

(1): the number of constraints; (2): the number of variables; (3): the number of nonzeros in A.

TABLE V Computational comparison between LPAKO and CPLEX

Problem	*LPAKO*	*CPLEK*	*Problem*	*LPAKO*	*CPLEX*
bnl2	5.27	3.38	maros-r7	54.91	23.07
25fv47	4.59	3.26	nesm	4.38	1.63
80bau3b	33.33	4.02	pilot	64.29	45.16
cycle	1.81	0.84	pilot87	194.78	145.17
czprob	1.31	0.45	pilot. ja	6.55	9.25
d2q06c	45.69	34.42	pilotnov	4.49	4.94
d6cube	40.50	37.64	scsd8	1.31	0.69
degen3	10.09	10.24	sctap3	1.44	0.43
fitld	1.59	0.66	shipo81	1.10	0.40
fitlp	1.95	0.36	shipl2l	2.30	0.60
fit2d	133.62	8.36	shipl2s	0.51	0.25
fit2p	119.25	46.77	stocfor3	138.72	18.45
greenbea	24.48	7.98	truss	21.21	11.51
geenbeb	17.52	5.53	wootlp	3.90	1.38
maros	1.77	0.73	woodw	6.42	1.12

Total running time.

machine (Sun UtraSparc 60, 768MB memory), and solved large-sized problems in NETLIB set [10]. These MPS-datafiles can be found in (1) and their characteristics are listed in Table IV. Computation times for each test problem taken by two programs are listed in Table V.

TABLE VI Results of presolving

Problem	*LPAKO*			*CPLEX*		
	(1)	*(2)*	*(3)*	*(1)*	*(2)*	*(3)*
bn12	1357 /	1313 /	10418	1381 /	1399 /	10236
25fv47	136 /	122 /	9986	138 /	124 /	9902
80bau3b	246 /	720 /	19813	297 /	1119 /	18981
cycle	780 /	784 /	14709	1284 /	1067 /	12944
czprob	406 /	634 /	5918	465 /	1083 /	4880
d2q06c	283 /	474 /	30859	316 /	606 /	30499
d6cube	13 /	2 /	37693	13 /	731 /	34270
degen3	93 /	93 /	24460	98 /	98 /	24420
fitld	0 /	0 /	13404	0 /	2 /	13386
fit1p	0 /	627 /	9241	0 /	250 /	9618
fit2d	0 /	0 /	129018	0 /	128 /	127841
fit2p	0 /	3000 /	47284	0 /	0 /	50284
greenbea	928 /	1093 /	29451	1380 /	2360 /	23025
greenbeb	927 /	1102 /	29142	1380 /	2369 /	22925
maros	190 /	239 /	8947	308 /	620 /	5746
maths-r7	984 /	1968 /	100486	984 /	2830 /	80167
nesm	56 /	273 /	12941	64 /	352 /	12883
pilot	96 /	303 /	40995	191 /	542 /	40189
pilot87	111 /	328 /	70465	144 /	392 /	70347
pilot. ja	181 /	538 /	11113	217 /	604 /	10900
pilotnov	195 /	436 /	11554	211 /	470 /	11448
scsd8	0 /	0 /	8584	0 /	0 /	8584
sctap3	134 /	713 /	7640	136 /	713 /	7630
ship081	98 /	24 /	12676	308 /	1184 /	7100
ship121	318 /	204 /	15504	542 /	1280 /	9222
ship12s	786 /	696 /	6108	884 /	916 /	4121
stocfor3	6096 /	4957 /	45212	8351 /	7349 /	48316
truss	0 /	0 /	27836	0 /	0 /	27836
woodlp	2 /	2 /	67708	74 /	866 /	44884
woodw	9 /	173 /	31332	543	4395 /	14536

(1): the number of deleted constraints; (2): the number of removed variables;
(3): the number of nonzero elements after presolving.

The above computational experiment shows that LPAKO runs much slower than CPLEX for some problems. We think the reason is twofold. One is that LPAKO does not have any special structure exploiting mechanism which can enhance the performance of solver greatly for special typed problems. The other is that the presolving of LPAKO is inferior to that of CPLEX. The performance of presolving of two programs are compared and the results are summarized in Table VI.

The lack of hardware exploiting programming techniques in LPAKO is another reason of its inferiority to CPLEX.

10 CONCLUDING REMARKS

LPAKO is a simplex-based LP program which can solve large, sparse LP problems with good speed and stability. It is written in the C language and its source code is available on the web site *http://orlab.snu.ac.kr/software/lpako/*. LPAKO has been widely used and will be improved continuously. Although the performance of LPAKO is poorer than commercial systems such as CPLEX, it outperforms almost all simplex-based LP systems in public domain. Thus we think that LPAKO is a very useful tool for researchers working in many areas of mathematical programming.

Acknowledgement

This report was supported by Grant No. 2000-1-31500-001-02 from the interdisciplinary research program of KOSEF.

References

[1] E.D. Andersen and K.D. Andersen (1995). Presolving in linear programming. *Mathematical Programming*, **71**, 221–245.
[2] R.H. Bartels and G.H. Golub (1969). The simplex method of linear programming using LU decomposition. *Communiradon of ACM*, **12**, 266–268.
[3] R.E. Bixby (1994). Progress in linear programming. *ORSA Journal on Computing*, **6**, 15–22.
[4] R.E. Bixby (1992). Implementing the simplex method: the initial basis. *ORSA Journal on Computing*, **4**, 267–284.
[5] R.E. Bixby and M.J. Saltzman (1994). Recovering an optimal LP basis from an interior point solution. *Operations Research Letters*, **15**, 169–178.
[6] I.S. Duff, A.M. Erisman and J.K. Reid (1986). *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford.
[7] S.C. Fang and S. Puthenpura (1993). *Linear Optimization and Extensions: Theory and Algorithms*. Prentice-Hall, Englewood Cliffs, N.J., USA.
[8] J.J. Forrest and D. Goldfarb (1992). Steepest-edge simplex algorithms for linear programming. *A Mathematical Programming*, **57**, 341–374.
[9] J.J. Forrest and J.A. Tomlin (1972). Updating triangular factors of the basis to maintain sparsity in the product-form simplex method. *Mathematical Programming*, **2**, 263–278.
[10] D.M. Gay (1985). Electronic mail distribution of linear programming test problems. *Mathematical Prograrmning Society Committee on Algorithms Newsletter*, **13**.
[11] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1989). A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, **45**, 437–474.
[12] D. Goldfarb and J.K. Reid (1973). A practical steepest-edge simplex algorithm. *Mathematical Programming*, **5**, 1–28.

[13] J. Gondzio (1997). Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, **9**(1), 73–91.
[14] P.M.J. Harris (1973). Pivot selection method of the Devex LP code. *Mathematical Programming*, **5**, 1–28.
[15] B. Jansen (1997). *Interior Point Techniques in Optimization: Complementary, Sensitivity and Algorithms*. Kluwer Academic Publishers, Dordrecht, Boston.
[16] N.K. Karmarkar (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, **4**(4), 373–395.
[17] S. Kim, J. Ahn, S. Lim and S. Park (2000). The design and development of command-based user interface for high speed linear programming solving system. *IE Interface*, **13**(4), 726–738 (in Korean).
[18] W. Kim, S. Lim and S. Park (1998). Efficient implementation of modified Forrest-Tomlin method in the simplex method. *Proceedings of KOR/MS '97 Autumn Conference*, 63–66.
[19] S. Lim, C. Park, W. Kim and S. Park (1997). An efficient implementation of one artificial variable technique in the simplex method. *Proceedings of KOR/MS'97 Autumn Conference*, 52–55.
[20] S. Lim, G. Kim and S. Park (2000). On the construction of initial basis in the simplex method. *Korean Management Science Review*, **17**(2), 13–20.
[21] S. Lim, M. Seong and S. Park (1999). An implementation of preprocessing for the simplex method. *Journal of the Korean Institute of Industrial Engineers*, **25**(2), 217–225.
[22] L.J. Lustig, R.E. Marsten and D.F. Shanno (1994). Interior point methods for linear programming: computational state of the art. *ORSA Journal on Computing*, **6**(1), 1–14.
[23] I. Maros and G. Mitra (1998). Strategies for creating advanced bases for large-scale linear programming problems. *INFORMS Journal on Computing*, **10**(2).
[24] N. Megiddo (1991). On finding primal- and dual-optimal bases. *ORSA Journal on Computing*, **3**(1).
[25] K.G. Murty (1983). *Linear Programming*. Wiley, New York.
[26] A. Nijemhuis and H.S. Wilf (1978). *Combinatorial Algorithms for Computers and Calculators*. Academic, New York.
[27] K. Noh, T. Seol, S. Lim and S. Park (2000). An experimental study on the scaling in linear programming. *Journal of Military Operations Research Society – Korea*. (submitted)
[28] W. Orchard-Hay (1968). *Advanced Linear-Programming Computing Techniques*. McGraw-Hill Book Company, New York, USA.
[29] S. Park (1999). *Linear Programming*. Minyong-sa, Seoul.
[30] C. Park, S. Lim, W. Kim and S. Park (1997). Implementation of the dynamic steepest edge method and anti-cycling method. *Proceedings of KOR/MS and KIIE '97 Spring Joint Conference*, 418–421.
[31] C. Park, S. Lim, W. Kim and S. Park (1998). Experimental comparisons of simplex method program's speed with various memory referencing techniques and data structures. *IE Interfaces*, **11**(2), 149–157 (in Korean).
[32] J.K. Reid (1975). A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. *Computer Science and System Devision*, CSS20, 1–23. A.E.RE., Harwell.
[33] C. Roos, T. Terlaky and J.-Ph. Vial (1997). *Theory and Algorithms for Linear Optimization*. John Wiley & Sons, New York, USA.
[34] Y. Seo and S. Park (1996). The study on the construction of initial basis in the simplex method. *Korean Management Science Review*, **13**(3), 105–113.
[35] H.D. Sherali, A.L. Soyster and S.G. Baines (1983). Nonadjacent extreme point methods for solving linear programs. *Naval Research Logistics Quarterly*, **30**, 145–161.

[36] U.H. Suhl and L.M. Suhl (1990). Computing sparse LU factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, **2**(4).
[37] L.M. Suhl and U.H. Suhl (1993). A fast LU update for linear programming. *Annals of Operations Research*, **43**, 33–47.
[38] T. Terlaky and S. Zhang (1993). Pivot rules for linear programming: a survey on recent theoretical developments. *Annals of Operations Research*, **46**, 203–233.
[39] P. Wolfe (1965). The composite simplex algorithm. *SIAM Review*, **7**, 42–54.
[40] S.J. Wright (1997). *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia.
[41] B. Yang and S. Park (1994). A sensitivity analysis on the cost coefficients for the Karmarkar's algorithm in linear programming. *Proceedings of AFORS'94*. Osaka.