Comput Optim Appl (2007) 37: 49–65
DOI 10.1007/s10589-007-9022-3

Progress in the dual simplex method for large scale LP problems: practical dual phase 1 algorithms

Achim Koberstein · Uwe H. Suhl

Received: 3 November 2004 / Revised: 16 February 2006 /
Published online: 14 March 2007
© Springer Science+Business Media, LLC 2007

Abstract The dual simplex algorithm has become a strong contender in solving large scale LP problems. One key problem of any dual simplex algorithm is to obtain a dual feasible basis as a starting point. We give an overview of methods which have been proposed in the literature and present new stable and efficient ways to combine them within a state-of-the-art optimization system for solving real world linear and mixed integer programs. Furthermore, we address implementation aspects and the connection between dual feasibility and LP-preprocessing. Computational results are given for a large set of large scale LP problems, which show our dual simplex implementation to be superior to the best existing research and open-source codes and competitive to the leading commercial code on many of our most difficult problem instances.

Keywords Dual simplex algorithm · Mathematical optimization system (MOPS) · Linear programming

1 Introduction

Lemke [18] developed the dual simplex method in 1954 but it was not found to be an alternative to the primal simplex method for nearly forty years. This changed in the early Nineties mainly due to the contributions of Forrest and Goldfarb [7] and Fourer [8]. During the last decade commercial solvers have made great progress in

A. Koberstein (✉)
Decision Support & Operations Research Lab, International Graduate School of Dynamic Intelligent Systems, University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany
e-mail: akober@upb.de

U.H. Suhl
Institut für Produktion, Wirtschaftsinformatik und Operations Research, Freie Universität Berlin, Garystraße 21, 14195 Berlin, Germany
e-mail: suhl@wiwiss.fu-berlin.de

establishing the dual simplex as a general solver for large-scale LP problems [1]. Nowadays, large scale LP problems can be solved either by an interior point, primal simplex or dual simplex algorithm or a combination of such algorithms. In practice, there are often LP-models, for which one of the three methods clearly outperforms the others. Despite of its success there are still only few publications in the research literature which evaluate the mathematical techniques proposed for the dual simplex algorithm within a state-of-the-art LP-system. In this paper, we investigate the task of obtaining a dual feasible basis as a starting point for the dual simplex algorithm from a computational perspective. Our analysis is based on the Mathematical OPtimization System (MOPS) (see [32, 33]), which has been deployed in numerous practical applications. It features efficient and stable implementations of the primal and the dual simplex algorithm, an interior point method, powerful LP and IP preprocessing and a branch-and-cut framework to solve linear and mixed-integer programs.

At first, we introduce a general version of the algorithm in Sect. 2, which incorporates the relatively new concept of a bound flipping ratio test. Then, we present practical versions of dual phase 1 methods, which have been proposed in the literature in Sect. 3. In Sect. 4 we address some aspects of LP preprocessing, which can drastically reduce the amount of dual infeasibility of a given starting bases. In Sect. 5 we mention some important implementation details to obtain a code which is both numerically stable and efficient on large scale problems. We give numerical results and analyse the performance of the different phase 1 methods and the overall performance of our code in Sect. 6. There, we also discuss the application of the dual simplex in a branch-and-bound framework to solve mixed-integer programs. Finally, we give a short conclusion in Sect. 7.

2 The dual simplex method for general linear programs

We consider linear programming problems (LP) of the form

$$\begin{aligned} \min\ & z = c^T x \\ \text{s.t.}\quad & Ax = b, \\ & l \le x \le u \end{aligned} \tag{1}$$

where A is a real $m \times n$ matrix of rank m and $m < n$. The entries of the vectors l and u may be minus or plus infinity, respectively. Let $\mathcal{J} = \{1, \ldots, n\}$ be the set of column indices. We denote by $\mathcal{J}^f = \{j \mid j \in \mathcal{J},\ l_j = -\infty \text{ and } u_j = \infty\}$ the set of free primal variables, by $\mathcal{J}^u = \{j \mid j \in \mathcal{J},\ l_j = -\infty \text{ and } u_j < \infty\}$ and $\mathcal{J}^l = \{j \mid j \in \mathcal{J},\ l_j > -\infty \text{ and } u_j = \infty\}$ sets of primal variables with one finite bound and by $\mathcal{J}^b = \{j \mid j \in \mathcal{J},\ l_j = -\infty \text{ and } u_j = \infty\}$ the set of primal variables with finite lower and upper bound. We call variables in $\mathcal{J}^b$ *boxed*. If for a variable $l_j = u_j = a$ for some $a \in \Re$, we call it *fixed*.

Using these definitions the dual problem associated with (1) can be stated as follows:

$$\begin{array}{lll} \max & Z = b^T\pi + \sum_{j\in J^l\cup J^b} l_j\upsilon_j + \sum_{j\in J^u\cup J^b} u_j\omega_j & \\ \text{s.t.} & A^T\pi + \upsilon + \omega = c, & \\ & \upsilon_j = 0, \quad \omega_j = 0 & \text{for all } j \in \mathcal{J}^f, \\ & \upsilon_j \geq 0, \quad \omega_j = 0 & \text{for all } j \in \mathcal{J}^l, \\ & \upsilon_j = 0, \quad \omega_j \leq 0 & \text{for all } j \in \mathcal{J}^u, \\ & \upsilon_j \geq 0, \quad \omega_j \leq 0 & \text{for all } j \in \mathcal{J}^b \end{array} \tag{2}$$

where the real vectors π, υ and ω have appropriate dimensions.

A *basis* $\mathcal{B} = \{k_1, \ldots, k_m\}$ is an ordered subset of $\mathcal{J}$, such that the submatrix $B = A_{\mathcal{B}}$ is nonsingular. The set of nonbasic column indices is denoted by $\mathcal{N} = \mathcal{J}\backslash\mathcal{B}$. A *primal basic solution* for a basis $\mathcal{B}$ is constructed by setting every primal nonbasic variable x_j, $j \in \mathcal{N}$, to one of its finite bounds (or to zero if it is free) and computing the primal basic variables as $x_{\mathcal{B}} = B^{-1}(b - A_{\mathcal{N}}x_{\mathcal{N}})$. $\mathcal{B}$ is called *primal feasible* if all primal basic variables are within their bounds, i.e., $l_j \leq x_j \leq u_j$ for all $j \in \mathcal{B}$. A *dual basic solution* for a basis $\mathcal{B}$ is constructed by computing dual multipliers $\pi = c_{\mathcal{B}}B^{-1}$ and reduced costs $d_{\mathcal{N}} = c_{\mathcal{N}} - A_{\mathcal{N}}^T\pi$ and setting $\upsilon_j = 0, \omega_j = 0$ if $j \in \mathcal{B}$, $\upsilon_j = d_j, \omega_j = 0$ if $j \in \mathcal{N}$ with $x_j = l_j$, and $\upsilon_j = 0, \omega_j = d_j$ if $j \in \mathcal{N}$ with $x_j = u_j$. If $j \in \mathcal{N}$ and $j \in \mathcal{J}^f$ (the associated primal variable is nonbasic and free) we set $\upsilon_j = d_j, \omega_j = 0$ if $d_j \geq 0$ and $\upsilon_j = 0, \omega_j = d_j$ if $d_j < 0$. $\mathcal{B}$ is called *dual feasible* if the dual slack variables υ and ω do not violate their respective zero bounds, i.e., $d_j \geq 0$ if $j \in \mathcal{N}$ with $x_j = l_j, d_j \leq 0$ if $j \in \mathcal{N}$ with $x_j = u_j$ and $d_j = 0$ if x_j is free. Note, that from the perspective of the primal problem dual infeasibilities can only occur at column indices associated with nonbasic primal variables. In particular, the dual slack variables associated with a nonbasic free primal variable can only be feasible if the corresponding reduced cost value is zero. The dual slack variables associated with a nonbasic fixed primal variable are always feasible. Infeasible dual slack variables associated with a nonbasic boxed primal variable x_j can always be made feasible by setting x_j to its other bound. Clearly, if such a bound switch is performed, the basic primal variables have to be updated.

The dual simplex method starts with a dual feasible basis and keeps changing the basis while the dual objective function can be improved and the problem does not turn out to be dual unbounded. We can state the algorithm as follows:

1. (Factor) Compute a factored representation of B^{-1},
   $x_{\mathcal{B}} = B^{-1}(b - A_{\mathcal{N}}x_{\mathcal{N}}), \pi = c_{\mathcal{B}}B^{-1}, d_{\mathcal{N}} = c_{\mathcal{N}} - A_{\mathcal{N}}^T\pi$.
2. (Pricing) Determine $p \in \mathcal{B}$ with $p = k_r$ and x_p infeasible.
   If $x_p < l_p$, set $\delta = x_p - l_p$. If $x_p > u_p$, set $\delta = x_p - u_p$.
   If $x_{\mathcal{B}}$ is feasible, then optimal. Go to 9.
3. (BTran) Compute $\rho_r = e_r^T B^{-1}$.
4. (Pivot row) Compute $\alpha^r = \rho_r A_{\mathcal{N}}$.
5. (Ratio test) If $x_p < l_p$, set $\underline{\alpha}^r = -\alpha^r$, o.w. set $\underline{\alpha}^r = \alpha^r$.
   Let $\mathcal{F} = \{j \in \mathcal{N}$: ($j$ free and $\underline{\alpha}_j^r \neq 0$) or ($x_j = l_j$ and
   $\underline{\alpha}_j^r > 0$) or ($x_j = u_j$ and $\underline{\alpha}_j^r < 0$)$\}$.

If $\mathcal{F} = \emptyset$, then dual unbounded. Go to 9.
Determine $q = \arg\min\{j \in \mathcal{F}\colon |d_j/\alpha_j^r|\}$ and $\theta^D = d_q/\alpha_q^r$.
6. (FTran) Compute $\alpha_q = B^{-1}a_q$.
7. (Basis change) Update Z: $Z := Z + \theta^D \delta$.
Update x: Compute $\theta^P = \delta/\alpha_q^r$.
Set $x_\mathcal{B} := x_\mathcal{B} - \theta^P \alpha_q$ and $x_q := x_q + \theta^P$.
Update $d_\mathcal{N}$: $d_j := d_j - \theta^D \alpha_j^r$ for $j \in \mathcal{N}$ and $d_p := -\theta^D$.
Update $\mathcal{B}, \mathcal{N}$: $\mathcal{B} := (\mathcal{B}\backslash\{p\}) \cup \{q\}$ and $\mathcal{N} := (\mathcal{N}\backslash\{q\}) \cup \{p\}$.
(LU-Update) Update the factored representation of B^{-1}.
8. If factor is requested, go to 1 otherwise go to 2.
9. End.

In step 1 the Basis is (re-)factorized. We use an improved version of the LU-factorization described in [34]. Also, the values of the primal basic variables and the reduced costs are (re-)computed. In step 2, which we call dual pricing, we determine an infeasible basic primal variable with index p that will leave the basis. Here, we use dual steepest edge pricing (see [7]). In steps 3 and 4 we compute the transformed pivot row α^r, which is used to determine an entering variable q in step 5. In step 6 we compute the transformed pivot column α_q, which is used to update the primal basic variables in step 7. The reduced costs, the objective function value, the basis and the LU-factorization is updated. For the FTran, BTran and LU-Update operations we use improved versions of the routines described in [35].

In our implementation we replace the simple ratio test in step 5 by a generalized version, which we call *bound flipping ratio test* (see [8, 9, 17, 19, 21]):

5.1. If $x_p < l_p$, set $\underline{\alpha}^r = -\alpha^r$, o.w. set $\underline{\alpha}^r = \alpha^r$.
Let $\mathcal{F} = \{j \in N\colon (j \text{ free and } \underline{\alpha}_j^r \neq 0)$ or $(x_j = l_j$ and $\underline{\alpha}_j^r > 0)$ or $(x_j = u_j$ and $\underline{\alpha}_j^r < 0)\}$ and $\mathcal{S} = \emptyset$.
5.2. If $\mathcal{F} = \emptyset$, then dual unbounded. Go to 9.
Determine $q = \arg\min\{j \in \mathcal{F}\colon |d_j/\alpha_j^r|\}$.
5.3. If q not boxed, go to 5.5
Else, if $x_q = l_q$ set $\theta_q^P = u_q - l_q$. If $x_q = u_q$ set $\theta_q^P = l_q - u_q$.
Set $\delta := \delta - \theta_q^P \alpha_q^r$.
5.4. If δ did not change sign, set $\mathcal{S} := \mathcal{S} \cup \{q\}$ and $\mathcal{F} := \mathcal{F}\backslash\{q\}$. Go to 5.2.
5.5. Else, flip bounds of variables in $\mathcal{S}$ and update $x_\mathcal{B}$:

$$x_\mathcal{B} := x_\mathcal{B} - B^{-1}\tilde{a} \quad \text{with } \tilde{a} = \sum_{j \in \mathcal{S}} \theta_j^P a_j.$$

The idea of this ratio test is to pass by breakpoints associated with boxed primal variables and flip their bounds to keep the associated dual slack variables feasible. This is done while the updated primal leaving variable is still infeasible.

3 Dual phase 1 methods

We refer to a technique or an algorithm which produces a dual feasible basis for an arbitrary LP (1) as a *dual phase* 1 *method*. The dual simplex method of the previous

section is called *dual phase* 2 because the basis remains dual feasible. The simplest method to provide a dual feasible basis that is long known (see for example [28]) for LPs in standard form (no upper bounds, nonnegative variables only) is to introduce an additional dual slack variable and punish its use in the dual objective function by some sufficiently large cost coefficient M. This is equivalent to adding an additional constraint of the form $\sum x_j \leq M$ to the primal problem. To apply this method to the general formulation (1) one could convert it to a problem in standard form by introducing additional variables and constraints. This would increase the size of the problem considerably. The other reason why this method is usually not used in practice is that a high value of M can lead to numerical problems and high iteration counts whereas a too small value might not produce a primal feasible solution. In the following, we only consider methods, which do not increase the size of the problem.

3.1 Artificial bounds

The *artificial bounds method* is a dual version of the *composite simplex method* which was originally developed for the primal simplex [36]. It can be seen as a one phase approach where infeasible dual slack variables are penalized in the dual objective function by a high cost coefficient M. From the perspective of the primal problem this means that infinite bounds of primal variables are replaced by artificial finite bounds.

Suppose some starting basis $\mathcal{B}$ is given and $\mathcal{N} = \mathcal{J} \backslash \mathcal{B}$. We can make dual slack variables associated with nonbasic boxed primal variables feasible by flipping primal bounds. Suppose, that the dual slack variables of the jth dual constraint ($j \in \mathcal{N}$) are infeasible, i.e., $d_j \leq 0, x_j$ at its lower bound and $u_j = \infty$. In this case an artificial bound $\underline{u}_j = M$ is introduced and x_j is set to $\underline{u}_j$ (d_j is not changed by this operation so the dual slack variables are now feasible). In the case $d_j \geq 0, x_j$ at upper bound and $l_j = -\infty$, an artificial bound $\underline{l}_j = -M$ is introduced and x_j is set to $\underline{l}_j$. $\mathcal{B}$ is dual feasible now, so we start dual phase 2 with one modification: when a variable with an artificial bound enters the basis, its original bounds are restored. If the dual phase 2 terminates with an optimal solution and no nonbasic variable is at an artificial bound, we are done. If there is a nonbasic primal variable x_j at an artificial bound, there are two possibilities: either M was chosen too small to remove all dual infeasibilities in the final basis or the problem is primal unbounded. To find out, we increase M (by multiplying with some constant) and start over. If M exceeds a certain threshold, we declare primal unboundedness. Our first implementation of the dual simplex was based on this approach. One disadvantage of this method is that we do not know how large to choose M. The right value for M depends strongly on the problem characteristic. If we choose it too small we risk many rounds of increasing and starting over. If we choose it too big we might encounter numerical problems. For this reason we decided not to include this method in our code.

3.2 Cost modifications, dual phase 2 and primal simplex

The basic idea of this method is similar to that of the artificial bounds method: making the starting basis dual feasible by modifying the problem formulation and restoring it

while executing dual phase 2. Here, additionally, we may need to deploy the primal simplex method at the end.

Given a starting basis $\mathcal{B}$ with $\mathcal{N} = \mathcal{J} \backslash \mathcal{B}$, dual slack variables associated with boxed primal variables are made feasible by flipping primal bounds. Then, for each remaining infeasible dual slack variable we shift the cost coefficient of the corresponding primal variable x_j with $j \in \mathcal{N}$ by $-d_j$, i.e., we set $\underline{c}_j = c_j - d_j$. This leads to a new reduced cost value $\underline{d}_j = \underline{c}_j - \pi^T a_j = c_j - d_j - \pi^T a_j = 0$, which is feasible for the dual. Note that by each cost shifting at the start of the method an additional dual degenerate position is created. Therefore, in order to reduce the danger of stalling during the further course of the algorithm, we perturb the cost values by a small margin ε, which is randomly generated in the interval $[10^{-6}, 10^{-5}]$. However, as our computational results will show, this procedure can only lessen the effects of the inherent additional degeneracy caused by this method, but not completely eliminate them. When the dual phase 2 terminates with a primal feasible basis, the original costs are restored. If the basis goes dual infeasible by this operation, we switch to the primal simplex method (primal phase 2).

This method is implemented in the LP code SoPlex, which was developed by Wunderling [37]. He reports that it yields good iteration counts and that for many problems it is not necessary to call the primal method at the end. This was not confirmed in our numerical tests though, where it was clearly outperformed by the other methods.

3.3 Pan's method

Pan proposed this method in [29] and further examined it computationally in [30]. The basic idea is to remove at least one dual infeasibility at every iteration without giving any guarantee that no new infeasibilities are created. This risk is minimized only by a proper, geometrically motivated selection of the leaving variable. We give the first description of the method for general linear programs (containing upper bounds). Before calling Pan's method we make dual slack variables associated with nonbasic boxed primal variables feasible by flipping primal bounds and do not consider them anymore in the further course of the algorithm. Then, the method proceeds as follows:

1. Let the set of indices associated with infeasible dual slack variables $Q = \{j \mid j \in \mathcal{N}\ (d_j < 0 \text{ and } x_j = l_j) \text{ or } (d_j > 0 \text{ and } x_j = u_j) \text{ or } (|d_j| > 0 \text{ and } j \text{ free with } x_j = 0)\}$. If Q is empty, the basis is dual feasible, go to 5. Select an entering index $q \in Q$ by some primal pricing rule. We use Dantzig's rule: $q = \arg\max\{|d_j| : j \in Q\}$.
2. Compute the transformed pivot column $\alpha_q = B^{-1} a_q$. If $d_q < 0$, set $\alpha_q := -\alpha_q$.
3. Determine a leaving index $p \in \mathcal{B}$, such that pth dual slack variables will be feasible after the basis change. Let $\mathcal{I} = \{i \mid i \in \{1, \ldots, m\}, j = k_i \text{ and } (l_j > -\infty \text{ and } \alpha_q^i < 0) \text{ or } (u_j < \infty \text{ and } \alpha_q^i > 0)\}$. If $\mathcal{I}$ is empty, the problem is dual infeasible, stop. Otherwise, select $p = \arg\max\{|\alpha_q^i| \mid i \in I\}$ and $k_r = p$. Compute $\theta^D = d_q / \alpha_q^r$.

4. Perform basis change as in step 7 of dual phase 2. To update $d_{\mathcal{N}}$, the transformed pivot row has to be computed in advance as in steps 3 and 4 of dual phase 2. Go to 1.
5. Call dual phase 2.

Note, that no ratio test is performed in this method, so we neither can guarantee a monotone reduction of the sum of dual infeasibilities nor of the number of dual infeasibilities. Also, there is no proof of convergence for this method. However, in our computational experiments it converges with a low iteration count on the vast majority of the tested instances. In that sense we can confirm Pan's results in [30]. On very few, mostly numerically difficult, instances it did not converge in an acceptable number of iterations. In that case, we switch to one of the other methods.

3.4 Minimizing the sum of dual infeasibilities

Although the basic idea of this method is long known it was recently reinvestigated in the literature by Fourer [8] and Maros [20]. The task of finding a basis with a minimal sum of dual infeasibilities can be formulated as an auxiliary LP problem, which both of the authors solve by a dual simplex type algorithm. They only differ in the way the bound flipping ratio test is applied: while Fourer only allows flips from *dual infeasible* to *dual feasible* Maros also allows the contrary, both under the precondition that the sum of dual infeasibilities decreases. To put it differently: Fourer's algorithm is monotone both in the sum and in the number of dual infeasibilities, while Maros's algorithm is monotone only in the sum of dual infeasibilities. Our version of the method is similar to the approach described in [17]. The basic idea is to directly apply the dual phase 2 to the auxiliary problem, which leads us to a dual phase 1 method that is equivalent to Maros' algorithm and allows for a more efficient and also simpler implementation.

Dual slack variables associated with boxed primal variables ($j \in \mathcal{J}^b$) are made feasible by flipping primal bounds. Then, we can state the problem of finding a basis with a minimal sum of dual infeasibilities as follows:

$$\begin{aligned} \max \quad & Z_0 = \sum_{j \in \mathcal{J}^l \cup \mathcal{J}^f, d_j < 0} d_j - \sum_{j \in \mathcal{J}^u \cup \mathcal{J}^f, d_j > 0} d_j \\ \text{s.t.} \quad & a_j^T \pi + d_j = c_j \quad \text{for all } j \in \mathcal{J}^l \cup \mathcal{J}^u \cup \mathcal{J}^f. \end{aligned} \tag{3}$$

Problem (3) is equivalent to the following formulation:

$$\begin{aligned} \max \quad & Z_0 = \sum_{j \in \mathcal{J}^l \cup \mathcal{J}^f} \omega_j - \sum_{j \in \mathcal{J}^u \cup \mathcal{J}^f} \upsilon_j \\ \text{s.t.} \quad & a_j^T \pi + \upsilon_j + \omega_j = c_j \quad \text{for all } j \in \mathcal{J}^l \cup \mathcal{J}^u \cup \mathcal{J}^f, \\ & \upsilon_j \geq 0, \\ & \omega_j \leq 0. \end{aligned} \tag{4}$$

The dual problem of problem (4) is:

$$\begin{aligned} \min \quad & z_0 = \sum_{j \in \mathcal{J}^l \cup \mathcal{J}^u \cup \mathcal{J}^f} c_j x_j \\ \text{s.t.} \quad & \sum_{j \in \mathcal{J}^l \cup \mathcal{J}^u \cup \mathcal{J}^f} a_j x_j = 0, \\ & -1 \le x_j \le 0 \quad \text{for all } j \in \mathcal{J}^u, \\ & 0 \le x_j \le 1 \quad \text{for all } j \in \mathcal{J}^l, \\ & -1 \le x_j \le 1 \quad \text{for all } j \in \mathcal{J}^f. \end{aligned} \tag{5}$$

Note, that problem (5) is a reduced version of our original problem (1) in the sense that it consists of a subset of the original set of columns and that bounds and right-hand side are changed. Since all variables of problem (5) are boxed every given starting basis can be made dual feasible by flipping primal bounds and the dual phase 2 algorithm given in Sect. 2 can directly be applied. If eventually it yields $z_0 = 0$, we have a dual feasible basis for our original problem and can start the dual phase 2 on problem (1). If $z_0 < 0$, the original problem is dual infeasible. In more detail, our version of the method is implemented as follows:

1. Mark indices related to boxed and fixed variables as not eligible.
2. Change the bounds of the remaining variables and the right-hand-side vector according to problem (5).
3. Start with an initial basis and make it dual feasible by flipping primal bounds.
4. Execute dual phase 2 on auxiliary problem (5).
5. If $z_0 < 0$, then original problem (1) is dual infeasible, stop.
6. If $z_0 = 0$, the current basis is dual feasible for problem (1).
7. Unmark indices related to boxed and fixed variables and restore original bounds and right-hand side.
8. Execute dual phase 2 on the original problem (1).

Fourer mentions in [8] that for model (3) the dual pricing step can be simplified: only those variables need to be considered in the selection process, which will become dual feasible after the basis change. Therefore, in our approach, only variables eligible to leave the basis are those, which go to their zero bound (being dual feasible in the original problem).

It is not difficult to see that applying the dual phase 2 (with bound flipping ratio test and the modified pricing) to the auxiliary problem (5) yields exactly the same algorithm as Maros describes in [20]. But from a computational point of view our approach seems to be more efficient since it is not necessary to expensively compute or update a special phase 1 pricing vector v as Maros does. It naturally corresponds to the primal basic solution $x_{\mathcal{B}}$ in the dual phase 2. Another advantage of our approach is, that no new code is needed to implement it besides the dual phase 2 code and every enhancement towards greater numerical stability and efficiency in the dual phase 2 code also improves the dual phase 1.

4 Preprocessing aspects for the dual simplex algorithm

It is widely recognized that LP preprocessing is very important for solving large-scale linear and integer optimization problems efficiently (see [4, 24]). This is true for both interior point and simplex algorithms. Although LP software and computers have become much faster, LP models have increased in size. Furthermore, LP optimizers are used in interactive applications and in integer programming where many LPs have to be solved. More efficient algorithms and improved implementation techniques are therefore still very important. Furthermore all practical LP/IP models are generated by computer programs either directly or within a modelling system. The model generator derives the computer model from the mathematical model structure and the model data. Most model generators have very limited capabilities for data analysis. As a consequence, there is usually a significant part of the model that is redundant. The main goals of LP preprocessing are:

- eliminate as many redundant constraints as possible
- fix as many variables as possible
- transform bounds of single structural variables (either tightening/relaxing them during LP preprocessing or tightening bounds during IP preprocessing)
- reduce the number of variables and constraints by eliminations

We refer here to the techniques described in [24]. The standard LP-preprocessing for LPs to be solved with an interior point or primal simplex algorithm uses bound tightening in an early phase of the LP-preprocessing. At the end of LP-preprocessing there is a reverse procedure where bounds are relaxed, i.e. redundant bounds are removed from the model.

As mentioned in previous sections of the paper, boxed variables play a key role in the dual simplex algorithm. Therefore tightened bounds are not relaxed if the dual simplex algorithm is used in our code. In Sect. 6 we show the impact of this strategy for some large scale LP problems.

5 Implementation aspects

In Sect. 2 we already mentioned some of the techniques we used to obtain an efficient and numerically stable implementation of the method, such as steepest edge pricing, bound flipping ratio test and LU-factorization and update.

Considering numerical stability and the reduction of degenerate iterations the implementation of the ratio test plays a key role. There, the following techniques have to be combined in an efficient way: 1. the bound flipping ratio test, 2. Harris' two pass ratio test [13], 3. cost shifting and dynamic cost perturbation (see [11] and [37]). Rather than using a priority queue to implement the bound flipping ratio test (as was recommended by Fourer and Maros) we basically stick to the simple search scheme given in Sect. 2. We do not explicitly keep a record of the flipped variables in the set $\mathcal{S}$ in the course of the ratio test. This is done later during the update of the reduced costs in step 7. Due to numerical inaccuracies and the use of Harris' ratio test with a tolerance of 10^{-7} nonbasic indices with slightly dual infeasible reduced costs d_j exist and can in fact be chosen as entering variables which would lead to small backward

steps in the objective function. To avoid the danger of numerical cycling and also achieve better behaviour on degenerate problems we do not accept backward or even zero steps and perform a small forward step in these cases by explicitly setting the dual step length $\theta^D = 10^{-12}$. Clearly, other nonbasic indices can go dual infeasible by this procedure so we shift their cost values such that their reduced costs are within the feasibility tolerance. Also, we shift the cost value of the entering variable such that its reduced cost is exactly zero.

Given a fast LU-factorization and update the exploitation of sparsity in the computation of the pivot row (step 4) and in the BTran and FTran operations and the use of proper data structures is crucial for efficiency. As pointed out earlier by other authors (see e.g. [2]), $\rho_r A_{\mathcal{N}}$ should be computed row-wise to allow for skipping zero entries in ρ_r. We actually use a row-wise compact storage of the whole constraint-matrix A, which turned out to be more efficient than updating a representation of $A_{\mathcal{N}}$ in every iteration, especially for problems with many columns. Also, we switch to column-wise computation if the density of ρ_r exceeds a certain threshold.

In each iteration of the method we have to solve four systems of linear equations: one BTran operation in step 3 and three FTran operations (transformed pivot column, update of $x_{\mathcal{B}}$ after bound flips, update of steepest edge weights). Using the LU-factorization of B, each of these operations involves the solution of two triangular systems of the form $H\gamma = h$, where H is one of the matrices L, L^T, U, U^T. In the cases, where h is very sparse (between one and ten entries, independent of model size) while γ is much denser—which is the normal situation—familiar sparsity-techniques are applied, such as row-wise and column-wise representations of H (see [22, p. 155f]). On bigger models and on some models with special structure, a situation called *hypersparsity* (cf. [12]) is more likely to occur, in which γ only has a couple of additional entries compared to h. Here, we apply a technique originally published by Gilbert and Peierls [10]: in a preceding symbolic phase a depth-first-search on the graph representation of H (which coincides with the column- or row-wise representation) is performed to compute an ordered list of the non-zero positions of γ. In the subsequent numerical phase, which is basically the same as in the normal case, we loop over this list only, avoiding all the zero-tests on entries of γ. Since there is no way of knowing the number of entries of γ in advance, we use simple estimates based on averages over preceding iterations to switch between the sparse and the hypersparse case whenever a triangular system is solved. For some models, this technique is indispensable to be able to solve them in a viable amount of time (see e.g. [23]).

Especially hypersparse models particularly benefit from the increased use of index stacks, which contain the nonzero positions of their associated vectors. In our implementation we develop an index stack for ρ_r during the BTran operation in step 3 and for α_q during the FTran operation in step 6. Thus, in the row-wise computation of the transformed pivot row α^r in step 4 and in the update of the primal basic solution $x_{\mathcal{B}}$ in step 7 we can loop solely over the nonzero positions avoiding the zero-tests on the rest of the vectors. In the case of ρ_r we also use the stack to zero-out the array for the next iteration. During the update of $x_{\mathcal{B}}$ we also exploit another observation: even if α_q is not very sparse, the number of positions of $x_{\mathcal{B}}$, which change their feasibility status, is usually small. For this reason we maintain an explicit list of its primal infeasible positions in the following way: if a position becomes infeasible which is not

on the list yet, its index is added and the square of its infeasibility value (for steepest edge pricing) is stored; if the position is already on the list, only its infeasibility value is updated. If a position becomes feasible, it is not removed from the list to avoid the linear search operation. After each refactorization the infeasibility list is recomputed. During the pricing in step 2 we only examine the positions contained in this list which are typically only a fraction of the total number of rows, especially towards the end of the optimization run.

6 Computational results

The computational results presented in this section are based on a set of one hundred test problems, which are taken from five different sources:

- 17 problems from the NetLib and Kennington test set [27].[1]
- 17 problems from the MipLib 2003 test set [25].[2]
- 21 problems from the Mittelmann test set [26].[3]
- 20 problems from the BPMPD test set [3].[4]
- 25 problems from our private collection of test problems [6].[5] The problems HAL_M_D, MUN1_M_D, MUN18_M_D and PTV15 are multi-depot bus scheduling models described in [15]. The instances P01–P20 are proprietary problems from various practical applications.

In general, we tried to find a composition of test problems, which mirrors the requirements in practical applications. Table 1 shows problem dimensions of five large problems from our test set, which were the most difficult w.r.t. Cplex 9.1 runtime.

Table 1 Problem dimensions of five large problems from our test set

Name	Source	Structurals	Constraints	Nonzeros
MUN1_M_D	[6]	1479833	163142	3031285
MUN18_M_D	[6]	675333	62148	1379406
RAIL4284	[26]	1092610	4284	11279748
STP3D	[25]	204880	159488	662128
STORMG2_1000	[26]	1259121	528185	3341696

[1] CRE-B, CRE-D, D2Q06C, DEGEN4, DFL001, FIT2P, GREENBEA, GREENBEB, KEN-13, KEN-18, MAROS-R7, OSA-30, OSA-60, PDS-10, PDS-20, PILOT, PILOT87.

[2] LP-relaxations of the problems: AIR04, ATLANTA-IP, DANO3MIP, DS, FAST0507, MOMENTUM2, MOMENTUM3, MSC98-IP, MZZV11, MZZV42Z, NET12, RD-RPLUSC-21, SEYMOUR, SP97AR, STP3D, T1717, VAN.

[3] BAXTER, DBIC1, FOME11, FOME20, FXM4_6, GEN4, LP22, MOD2, NSCT2, PDS-100, PDS-40, QAP12, RAIL4284, RAIL507, SELF, SGPF5Y6, STORMG2_1000, STORMG2-125, WATSON_1, WATSON_2, WORLD.

[4] AA3, BAS1LP, CO9, CQ9, DBIR1, EX3STA1, JENDREC1, LPL1, LPL3, MODEL10, NEMSPMM2, NEMSWRLD, NUG08, NUG12, RAT5, SCFXM1-2R-256, SLPTSK, SOUTH31, T0331-4L, ULEVIMIN.

[5] FA, HAL_M_D, MUN1_M_D, MUN18_M_D, P01, P02, P03, P04, P05, P06, P07, P08, P09, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, PTV15.

For the other problems, dimensions and detailed individual benchmarking results can be found in [16]. In the following we will analyse and interpret our benchmarking data by means of the performance measures sum, geometric mean and performance profiles (cf. [5]).

From the dual phase 1 methods presented in Sect. 3, we implemented the subproblem approach of minimizing the sum of the dual infeasibilities (SP), the method by cost modification (CM) and Pan's method. As mentioned before Pan's method does not provide a theoretical convergence guarantee, which turned out to pose a problem on a handful of mostly numerically difficult problems. Therefore we provided our Pan code with a simple checking rule, which switches to the subproblem phase 1, if the number of dual infeasibilities does not decrease adequately. This variant will be denoted by Pan + SP.

We also tried two versions of the CM method: in the first version we restored the original cost coefficients after each refactorization, if the corresponding reduced cost values stay dual feasible, in the second version we do not touch the modified cost vector until the end of the dual phase 2. Here, we will present results only for the first variant, since it worked clearly better (as expected) than the second one.

In our first experiment we conducted benchmarks for the four different methods on those 46 problems[6] of our test set, for which the starting (all-logical) basis is dual infeasible and cannot be made dual feasible by pure bound flipping. These test runs were conducted under Windows XP Professional on a standard Intel Pentium IV PC with 3.2 GHz and 1 GB of main memory. Our code was compiled with Compaq Visual Fortran Compiler V6.6. Table 2 summarizes the results by listing sums and geometric means (where appropriate) for runtime and iteration counts. Pan's method is not included since it failed on three problems to achieve a dual feasible basis. Figure 1 shows a performance profile over runtime for all of the four methods.

Each of the four algorithms performs quite well on about two thirds of the test instances compared to the respective best method, with a slight edge for Pan + SP.

Table 2 Summary of benchmark results for dual phase 1 methods

	Pan + SP	SP	CM
Sum CPU time (sec)	8521.1	9254.9	11625.8
Geom. mean CPU time	30.1	31.3	32.7
Sum total iters	2093129	2160389	2183591
Geom. mean total iters	18602.6	18910.7	19729.6
Sum degen iters[a]	260277	308879	649566
Sum phase 1 iters	432107	461127	–
Geom. mean phase 1 Iters	164.9	195.1	–
Sum dual iters	2079179	2146392	1676332
Sum primal iters[b]	13950	13997	507259

[a]We consider an iteration as degenerate, if $\theta^D < 10^{-7}$.

[b]The primal simplex is generally used at the end of the dual phase 2 to remove dual infeasibilities caused by restoring the original cost vector after cost perturbation and shiftings.

[6]CO9, CQ9, CRE-B, CRE-D, D2Q06C, DBIC1, DEGEN4, DFL001, EX3STA1, FOME11, FOME12, FOME13, FXM4_6, GREENBEB, JENDREC1, MOD2, MODEL10, MZZV11, NEMSPMM2, NET12, P02, P04, P05, P06, P07, P09, P10, P11, P12, P13, P14, P19, P20, PILOT, PILOT87, RD-RPLUSC-21, SEYMOUR, SGPF5Y6, SLPTSK, SOUTH31, STORMG2_1000, STORMG2-125, ULEVIMIN, WATSON_1, WATSON_2, WORLD.

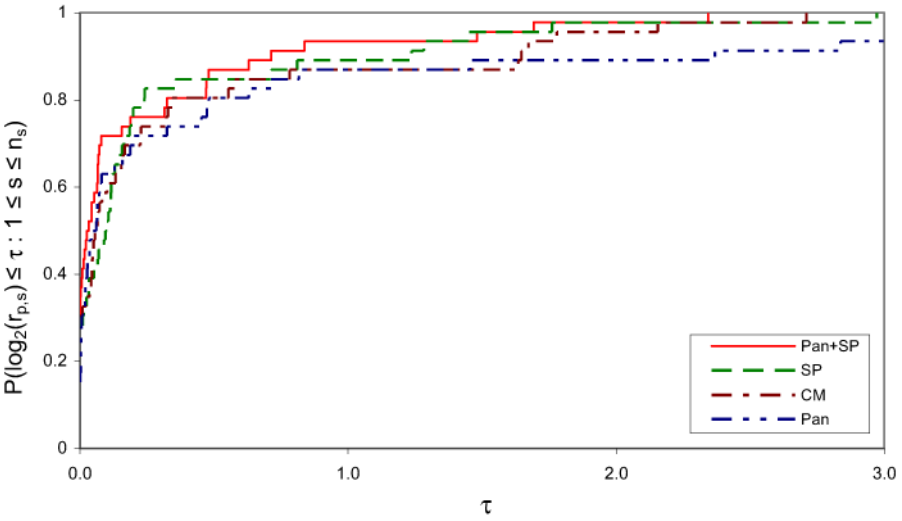

Fig. 1 Performance profile over phase 1 test set: solution time using four different dual phase 1 methods

On roughly 20% of the problems the original Pan method and the CM method show significant difficulties. For the CM method these are mainly those problems, which need a large proportion of primal simplex[7] iterations (WATSON_1, WATSON_2, DEGEN4, ULEVIMIN). An exception is the problem DBIC1, on which the primal simplex seems to be superior to the dual simplex. It can be seen as a fundamental drawback of this method, that it is basically unpredictable how many iterations are performed by the dual simplex and how many are performed by the primal simplex method for a given problem. For our test set, almost 25% of the iterations were in primal simplex. This makes it almost impossible for the user to choose the right solution algorithm for his LP model. Furthermore, this method has an inherent tendency to increase the number of degenerate iterations. *Pan* has problems on numerically difficult instances like P13, P14, P19 and P20. On these problems, the SP method works significantly better. To summarize we can say, that the combined method Pan + SP has the best overall performance, with a slight edge compared to SP. Therefore the Pan + SP method is used by default in our dual simplex code and was also deployed in the remaining computational studies of this section.

In a second experiment we investigate the impact of the treatment of the primal bounds after LP preprocessing. Two variants are compared on the complete problem test set: the first one keeps the reduced bounds, the second expands the bounds after LP preprocessing. As above the test runs were conducted under Windows XP Professional on a standard Intel Pentium IV PC with 3.2 GHz and 1 GB of main memory, but here we use an executable generated by the Intel Visual Fortran Compiler V8.0, which turned out to be superior to the Compaq version. The results are summarized in Table 3 and the performance profile in Fig. 2.

[7]The primal simplex code in MOPS is inferior to the dual simplex code in particular on large problems, since it still lacks some important implementation techniques (e.g. hypersparsity). But even with an improved primal code the dual simplex is generally seen as superior to the primal simplex (cf. [1]).

Table 3 Summary of benchmark results with reduced and expanded bounds after LP preprocessing

	Reduced bounds	Expanded bounds
Sum CPU time (sec)	38816.5	39643.8
Geom. mean CPU time	23.0	25.6
Sum total iters	3816302	3971111
Sum degen. iters	446812	460097
Sum dual phase 1 iters	433351	497465
Number of problems with dual phase 1	46	52

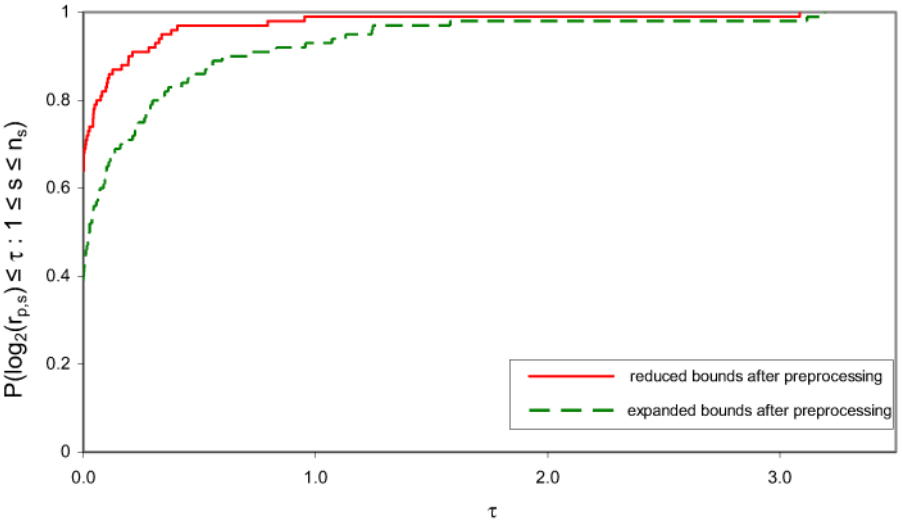

Fig. 2 Performance profile over all test models: solution time with reduced and expanded bounds after LP preprocessing

The performance profile shows, that keeping the reduced bounds after LP preprocessing clearly improves the overall performance of the dual simplex method. The number of dual phase 1 iterations is significantly reduced for many models, for some of the models the execution of a dual phase 1 method even becomes superfluous. This is not surprising, since additional finite bounds tend to increase the number of boxed variables, which are made dual feasible by feasibility correction. Furthermore, the number of degenerate iterations decreases compared to the variant with expanded bounds. Tighter bounds probably increase the impact of the bound flipping ratio test, which can be seen as an anti-degeneracy technique.

To evaluate the performance of the MOPS dual simplex implementation we compared it to the dual simplex codes of the LP-systems Soplex 1.2.1 [31],[8] COIN LP

[8]For Soplex, we used the entering algorithm, which conceptually can be seen as the primal simplex method using the dual simplex method to achieve primal feasibility at the start. In practice however, three fourths of

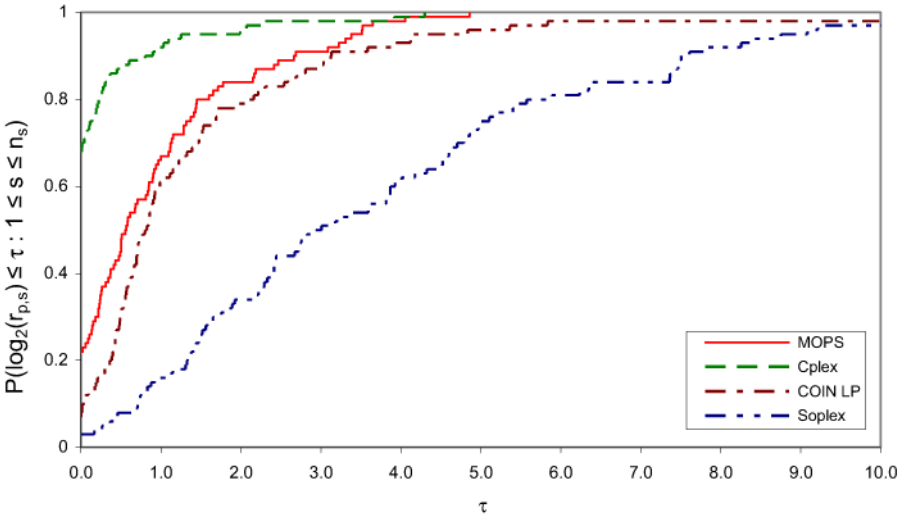

Fig. 3 Performance profile comparing dual simplex codes of four different LP systems

Table 4 Solution times (seconds CPU time) for five problems from our test set, which were the most difficult w.r.t. Cplex 9.1 runtime (cf. Table 1)

Problem	MOPS 7.9 CPU time	Soplex 1.2.1 CPU time	Clp 1.02.02 CPU time	Cplex 9.1 CPU time
MUN1_M_D	15720.8	>72000	12940.0	14295.8
MUN18_M_D	3142.3	>72000	3309.2	5469.3
RAIL4284	5944.9	10058.1	7417.0	4019.8
STP3D	2769.1	6726.7	1253.0	1021.2
STORMG2_1000	3464.4	15550.5	1747.4	537.9

(CLP) 1.02.02 and Cplex 9.1 [14] with default settings and the respective system-specific LP preprocessing. The test runs were conducted under Windows XP Professional on a standard Intel Pentium IV PC with 3.2 GHz and 1 GB of main memory.

The MOPS executable was built by the Intel Visual Fortran Compiler V8.0, the Soplex and CLP code was compiled using Microsoft Visual C++ V6.0. Figure 3 shows the resulting performance profile over solution time and Table 4 contains individual results for five of the most difficult test problems.

MOPS clearly outperforms Soplex, which fails on two of the largest problems (MUN1_M_D and MUN18_M_D).[9] On about 20% of the test models, MOPS achieves the best solution time. However, on this benchmark, Cplex 9.1 clearly yields the best overall performance. We want to emphasise again, that neither the techniques

the iterations (average, may vary tremendously on different problems) are performed by the dual simplex part of the code.

[9] Soplex exceeds the 12 hour time limit.

used in the COIN LP code nor the internals of the Cplex code are documented in the research literature.

7 Conclusion

In this paper we addressed the problem of finding a dual feasible basic solution for a general LP problem from a computational point of view. We presented practical versions of dual phase 1 methods that have been proposed in the literature and showed how to implement them efficiently. The algorithm of Pan combined with other techniques showed the best overall results. Furthermore, we demonstrated the impact of LP preprocessing and a simple bound tightening technique on dual feasibility and the overall performance of the dual simplex algorithm. Computational results were given on a large set of large scale LP problems, which showed our dual simplex implementation to be superior to the best existing research and open-source codes and competitive to the leading commercial code on many of our most difficult problem instances.

References

1. Bixby, R.E.: Solving real-world linear programs: a decade and more of progress. Oper. Res. **50**(1), 3–15 (2002)
2. Bixby, R.E., Martin, A.: Parallelizing the dual simplex method. Informs J. Comput. **12**(1), 45–56 (2000)
3. BPMPD test problems. http://www.sztaki.hu/~meszaros/bpmpd/
4. Brearley, A.L., Mitra, G., Williams, H.P.: Analysis of mathematical programming problems prior to applying the simplex algorithm. Math. Program. **15**, 54–83 (1975)
5. Dolan, E.D., More, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
6. DSOR test problems. http://dsor.upb.de/koberstein/lptestset/
7. Forrest, J.J., Goldfarb, D.: Steepest edge simplex algorithms for linear programming. Math. Program. **57**(3), 341–374 (1992)
8. Fourer, R.: Notes on the dual simplex method. Draft report (1994)
9. Gabasov, R., Kirillova, F.M., Kostyukova, O.I.: A method of solving general linear programming problems. Doklady AN BSSR **23**(3), 197–200 (1979) (in Russian)
10. Gilbert, J.R., Peierls, T.: Sparse partial pivoting in time proportional to arithmetic operations. SIAM J. Sci. Stat. Comput. **9**, 862–874 (1988)
11. Gill, P., Murray, W., Saunders, M., Wright, M.: A practical anti-cycling procedure for linearly constrained optimization. Math. Program. **45**, 437–474 (1989)
12. Hall, J.A.J., Mc Kinnon, K.I.M.: Hyper-sparsity in the revised simplex method and how to exploit it. Comput. Math. Appl. (2005, to appear)
13. Harris, P.: Pivot selection method of the Devex LP code. Math. Program. **5**, 1–28 (1973)
14. ILOG: Cplex 9.0 reference manual (2003)
15. Kliewer, N.: Optimierung des Fahrzeugeinsatzes im öffentlichen Personennahverkehr: Modelle, Methoden und praktische Anwendungen. Dissertation at the University of Paderborn II, Fakultät für Wirtschaftswissenschaften, Department Wirtschaftsinformatik (2005), http://ubdata.uni-paderborn.de/ediss/05/2005/kliewer/
16. Koberstein, A.: The Dual Simplex Method: Techniques for a fast and stable implementation. Dissertation at the University of Paderborn II, Fakultät für Wirtschaftswissenschaften, Department Wirtschaftsinformatik (2005)
17. Kostina, E.: The long step rule in the bounded-variable dual simplex method: numerical experiments. Math. Methods Oper. Res. **55**, 413–429 (2002)

18. Lemke, C.E.: The dual method of solving the linear programming problem. Nav. Res. Log. Q. **1**, 36–47 (1954)
19. Maros, I.: A piecewise linear dual procedure in mixed integer programming. In: Giannesi et al. (eds.) New Trends in Mathematical Programming, pp. 159–170. Kluwer, Dordrecht (1998)
20. Maros, I.: A piecewise linear dual phase-1 algorithm for the simplex method with all types of variables. Comput. Optim. Appl. **26**, 63–81 (2003)
21. Maros, I.: A generalized dual phase-2 simplex algorithm. Eur. J. Oper. Res. **149**(1), 1–16 (2003)
22. Maros, I.: Computational Techniques of the Simplex Method. Kluwer International Series. Kluwer, Dordrecht (2003) ISBN 1-4020-7332-1
23. McBride, R.D., Mamer, J.W.: Solving multicommodity flow problems with a primal embedded network simplex algorithm. INFORMS J. Comput. **9**(2), 154–163 (1997)
24. Mészáros, C., Suhl, U.H.: Advanced preprocessing techniques for linear and quadratic programming. OR Spectrum **25**(4), 575–595 (2003)
25. MipLib 2003 test problems. http://miplib.zib.de/
26. Mittelmann test problems. ftp://plato.asu.edu/pub/lpfree.html/
27. NetLib test problems. http://www.netlib.org/lp/data/
28. Padberg, M.W.: Linear Optimization and Extensions. Springer, Berlin (1995)
29. Pan, P.Q.: Practical finite pivoting rules for the simplex method. OR Spektrum **12**, 219–225 (1990)
30. Pan, P.Q.: The most-obtuse-angle row pivot rule for achieving dual feasibility: a computational study. Eur. J. Oper. Res. **101**(1), 167–176 (1997)
31. Sequential object-oriented simplex. http://www.zib.de/Optimization/Software/Soplex/
32. Suhl, U.H.: MOPS—Mathematical optimization system. Eur. J. Oper. Res. **72**, 312–322 (1994)
33. Suhl, U.H.: MOPS home page. World Wide Web, http://www.mops-optimizer.com/ (1999)
34. Suhl, U.H., Suhl, L.M.: Computing sparse LU factorizations for large-scale linear programming bases. ORSA J. Comput. **2**, 325–335 (1990)
35. Suhl, L.M., Suhl, U.H.: A fast LU-update for linear programming. Ann. Oper. Res. **43**, 33–47 (1993)
36. Wolfe, P.: The composite simplex algorithm. SIAM Rev. **7**(1), 42–54 (1965)
37. Wunderling, R.: Paralleler und objektorientierter simplex. Techniqual report TR-96-09, Konrad-Zuse-Zentrum für Informationstechnik, Berlin (1996)