Computational study of the GDPO dual phase-1 algorithm

István Maros*
Department of Computing, Imperial College, London
Email: i.maros@ic.ac.uk

Departmental Technical Report 2006/6
ISSN 1469–4174

July 2006

Abstract

Maros's GDPO algorithm for phase-1 of the dual simplex method possesses some theoretical features that have potentially huge computational advantages. This paper gives account of a computational analysis of GDPO. Experience of a systematic study involving 48 problems shows that the predicted performance advantages can materialize to a large extent making GDPO an indispensable tool for dual phase-1.

1 Introduction

The simplex method has two main versions: the primal and the dual simplex algorithm. Since Dantzig's seminal work [1] in 1951, the primal version received far more attention than Lemke's dual [6] from 1954. As a result of this "bias" primal based simplex implementations have evolved continuously and can solve large linear programming (LP) problems reliable and efficiently. The dual simplex did not follow suit and its use was limited to cases where a dual feasible basis was available, like a simple *Branch and Bound* (B&B) type solution of mixed integer linear programming (MILP) problems using dual phase-2. Recently, dual phase-2 has undergone a substantial progress so that it can handle all types of variables algorithmically [7; 9; 4] and it is very effective in practice.

The newly emerging methods for MILP use local techniques at the nodes of the search tree like *logical testing, implied bounds, added cuts*. In such cases it is not true anymore that the optimal basis of a parent node is dual feasible for the child nodes. Therefore, dual must start in phase-1. This necessitates the development of an efficient dual phase-1 algorithm. Another motivation for a new dual phase-1 algorithm was to make dual a competitive alternative to the primal for general LP problems. The result of the author's ensuing work was the creation of the GDPO (Generalized Dual Phase One) algorithm [8; 9]. This algorithm possesses some interesting theoretical features that have potentially huge computational advantages. The extent of the advantages has not been known. Therefore, to see how the algorithm works in practice the author has conducted a systematic computational study of GDPO. Experience on 48 problems shows that the theoretically proven advantages of the algorithm (discussed in detail in [8] and [9]) can materialize in practice to a really large extent. This paper gives account of the study.

*This work was partly supported by EPSRC grant EP/C533461/1

The rest of the paper is organized in the following way. Section 2 gives the general form of the LP problem followed by section 3 with the brief theoretical description of GDPO. Section 4 is devoted to the computational analysis of GDPO, while section 5 gives a summary of our findings.

2 Problem statement

Consider the following primal linear programming (LP) problem:

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x}, \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \end{array} \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{c}$, $\mathbf{x}$, $\boldsymbol{\ell}$ and $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$. Some or all of the components of $\boldsymbol{\ell}$ and $\mathbf{u}$ can be $-\infty$ or $+\infty$, respectively. $\mathbf{A}$ itself is assumed to contain a unit matrix $\mathbf{I}$, that is, $\mathbf{A} = [\mathbf{I}, \bar{\mathbf{A}}]$, so it is of full row rank. Variables which multiply columns of $\mathbf{I}$ transform every constraint to an equation and are often referred to as *logical variables*. Variables which multiply columns of $\bar{\mathbf{A}}$ are called *structural variables*.

By some elementary transformations it can be achieved that all variables (whether logical or structural) fall into four categories as shown in Table 1. Note, type-1 logical variables correspond to range constraints. More details of problem statement can be found in [9].

Table 1: Types of variables

Feasibility range	Type	Reference
$x_j = 0$	0	Fixed variable
$0 \le x_j \le u_j < +\infty$	1	Bounded variable
$0 \le x_j \le +\infty$	2	Non-negative variable
$-\infty \le x_j \le +\infty$	3	Free variable

2.1 The dual problem

First, we restate the primal problem to contain bounded variables only.

$$(P1) \quad \begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x}, \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{0} \le \mathbf{x} \le \mathbf{u}, \end{array}$$

where all components of $\mathbf{u}$ are finite.

A basis to (P1) is denoted by $\mathbf{B}$ and is assumed (without loss of generality) to be the first m columns. Thus, $\mathbf{A}$ is partitioned as $\mathbf{A} = [\mathbf{B}, \mathbf{R}]$, with $\mathbf{R}$ denoting the nonbasic part of $\mathbf{A}$. The components of $\mathbf{x}$ and $\mathbf{c}$ are partitioned accordingly. Column j of $\mathbf{A}$ is denoted by $\mathbf{a}_j$. A basic solution to (P1) is

$$\mathbf{x}_{\mathcal{B}} = \mathbf{B}^{-1}\left(\mathbf{b} - \sum_{j\in\mathcal{U}} u_j \mathbf{a}_j\right),$$

where $\mathcal{U}$ is the index set of nonbasic variables at upper bound. The ith basic variable is denoted by x_{Bi}. The d_j reduced cost of variable j is defined as $d_j = c_j - \boldsymbol{\pi}^T\mathbf{a}_j = c_j - \mathbf{c}_{\mathbf{B}}^T\mathbf{B}^{-1}\mathbf{a}_j$ which is further equal to $c_j - \mathbf{c}_B^T\boldsymbol{\alpha}_j$ if the notation $\boldsymbol{\alpha}_j = \mathbf{B}^{-1}\mathbf{a}_j$ is used. $\mathbf{d}$ is the vector of reduced costs.

The dual of (P1) is:

$$(D1) \quad \begin{array}{ll} \text{maximize} & \mathbf{b}^T\mathbf{y} - \mathbf{u}^T\mathbf{w}, \\ \text{subject to} & \mathbf{A}^T\mathbf{y} - \mathbf{w} \le \mathbf{c}, \\ & \mathbf{w} \ge \mathbf{0}, \end{array}$$

where $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{w} \in \mathbb{R}^n$ are the dual variables. It is to be noted that the y variables are unrestricted in sign (free variables).

If only type-2 variables are present in (P1) then we obtain

$$(\text{P2}) \quad \begin{array}{ll} \min & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \ge \mathbf{0}, \end{array}$$

and its dual is

$$(\text{D2}) \quad \begin{array}{ll} \max & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} & \mathbf{A}^T\mathbf{y} \le \mathbf{c}. \end{array}$$

Let us consider the (P2)$-$(D2) pair. Note, $\mathbf{A}$ contains a unit matrix and $m < n$. With the introduction of vector $\mathbf{w} = [w_1, \ldots, w_n]^T$ of dual logical variables (D2) can be rewritten as

$$\max \quad \mathbf{b}^T\mathbf{y} \tag{2}$$

$$\text{s.t.} \quad \mathbf{A}^T\mathbf{y} + \mathbf{w} = \mathbf{c}, \tag{3}$$

$$\mathbf{w} \ge \mathbf{0}. \tag{4}$$

Let $\mathbf{B}$ be a basis to $\mathbf{A}$. It need not be primal feasible. Rearranging (3), we get $\mathbf{w}^T = \mathbf{c}^T - \mathbf{y}^T\mathbf{A}$, or in partitioned form

$$\mathbf{w}_{\mathcal{B}}^T = \mathbf{c}_{\mathcal{B}}^T - \mathbf{y}^T\mathbf{B}, \tag{5}$$

$$\mathbf{w}_{\mathcal{R}}^T = \mathbf{c}_{\mathcal{R}}^T - \mathbf{y}^T\mathbf{R}. \tag{6}$$

The nonnegativity (and, in this case, the feasibility) requirement (4) of $\mathbf{w}$ in partitioned form is $[\mathbf{w}_{\mathcal{B}}^T, \mathbf{w}_{\mathcal{R}}^T]^T \ge \mathbf{0}$. Choosing $\mathbf{y}^T = \mathbf{c}_{\mathcal{B}}^T\mathbf{B}^{-1}$ we obtain

$$\mathbf{w}_{\mathcal{B}}^T = \mathbf{c}_{\mathcal{B}}^T - \mathbf{c}_{\mathcal{B}}^T\mathbf{B}^{-1}\mathbf{B} = \mathbf{0}, \tag{7}$$

$$\mathbf{w}_{\mathcal{R}}^T = \mathbf{c}_{\mathcal{R}}^T - \mathbf{c}_{\mathcal{B}}^T\mathbf{B}^{-1}\mathbf{R} = \mathbf{d}_{\mathcal{R}}^T \ge \mathbf{0}, \tag{8}$$

where $\mathbf{d}_{\mathcal{R}}$ denotes the vector formed by the primal reduced costs of the nonbasic variables. Since (7) is satisfied with any basis and $\mathbf{y}$ is unrestricted in sign, a basis $\mathbf{B}$ is dual feasible if it satisfies (8). This is, however, nothing but the primal optimality condition. Therefore, we can conclude that the dual feasibility condition is equivalent to the primal optimality condition. Additionally, the dual logicals are equal to the primal reduced costs. Therefore, w_j and d_j can be used interchangeably.

Stating the dual when the primal has all types of variables is cumbersome. However, we can think of the reduced costs of the primal problem as the logical variables of the dual, c.f. [9]. In this way dual feasibility can be expressed quite simply as shown in the next section.

In practice, dual algorithms work on the primal problem using the computational tools of the sparse primal simplex method (SSX) but perform basis changes according to the rules of the dual.

The creation of the updated pivot row p, i.e., the computation of α_j^p for all nonbasic indices j is an expensive operation in SSX (c.f. [10]). Traditional dual methods based on the Dantzig type pivot selection make *one iteration* with the pivot row and discard it. The possible multiple use of this row has motivated the author to develop a new algorithm called GDPO [8]. GDPO makes *one step* with the pivot row which, however, can correspond to many iterations of the traditional method with very little extra work. As GDPO is monotone only in the sum of infeasibilities it has an increased flexibility. It also has some additional favorable features that enhance its effectiveness and efficiency.

3 The GDPO algorithm

This section gives a brief description of GDPO. A more detailed discussion can be found in the original paper by Maros [8].

3.1 Theoretical background

It is known that the primal reduced costs are the same as the dual logical variables, denoted by d_j (c.f. [9, pages 261–262]). Therefore, the feasible solutions of the dual of (1) satisfy the following conditions.

Type(x_j)	Value	d_j	Remark
0	$x_j = 0$	Immaterial	
1	$x_j = 0$	≥ 0	
1	$x_j = u_j$	≤ 0	$j \in \mathcal{U}$
2	$x_j = 0$	≥ 0	
3	$x_j = 0$	$= 0$	

(9)

In other words, a dual solution defined by sets $(\mathcal{B},\mathcal{U})$ is feasible if the corresponding d_j values satisfy (9).

Since d_j of a type-0 variable is always feasible such variables can be, and in fact are, ignored in dual phase-1. Furthermore, dual logicals of type-1 (bounded) variables can easily be made feasible by moving the corresponding primal variables to their opposite bound. It can be done without basis change by simply updating the primal basic solution. For details, see [8] where this operation is called *feasibility correction*.

It can be concluded that only type-2 and type-3 variables need to be considered in an algorithm for dual feasibility. We define two infeasibility sets for them as follows.

$$\mathcal{P} = \{j : d_j > 0 \text{ and type}(x_j) = 3\}, \tag{10}$$

and

$$\mathcal{M} = \{j : d_j < 0 \text{ and type}(x_j) \geq 2\}. \tag{11}$$

If all variables are of type-1 any basis can be made dual feasible by feasibility correction.

Using infeasibility sets of (10) and (11), the sum of dual infeasibilities is defined as

$$f = \sum_{j \in \mathcal{M}} d_j - \sum_{j \in \mathcal{P}} d_j, \tag{12}$$

where any of the sums is zero if the corresponding index set is empty. It is always true that $f \leq 0$. In dual phase-1 the objective is to maximize f subject to the dual feasibility constraints. When $f = 0$ is reached the solution becomes dual feasible (maybe after a feasibility correction). If it cannot be achieved the dual is infeasible.

In an iteration of the dual simplex method first the outgoing basic variable is selected which defines the pivot row. Let us assume row p is selected somehow (i.e., the pth basic variable x_{Bp} will leave the basis). The elimination step of the simplex transformation subtracts some multiple of row p from $\mathbf{d}_{\mathcal{R}}$. If this multiplier is denoted by t the transformed value of each d_j can be written as a function of t:

$$d_j(t) = d_j - t\alpha_j^p, \quad j \in \mathcal{R}. \tag{13}$$

With this notation, $d_j(0) = d_j$ and the sum of infeasibilities as a function of t can be expressed (assuming t is small enough such that $\mathcal{M}$ and $\mathcal{P}$ remain unchanged) as:

$$f(t) = \sum_{j\in\mathcal{M}} d_j(t) - \sum_{j\in\mathcal{P}} d_j(t) = f(0) - t\left(\sum_{j\in\mathcal{M}} \alpha_j^p - \sum_{j\in\mathcal{P}} \alpha_j^p\right). \tag{14}$$

Clearly, f of (12) can be obtained as $f = f(0)$.

The change in the sum of dual infeasibilities, if t moves away from 0, is:

$$\Delta f = f(t) - f(0) = -t\left(\sum_{j\in\mathcal{M}} \alpha_j^p - \sum_{j\in\mathcal{P}} \alpha_j^p\right). \tag{15}$$

Introducing notation

$$v_p = \sum_{j\in\mathcal{M}} \alpha_j^p - \sum_{j\in\mathcal{P}} \alpha_j^p \tag{16}$$

(15) can be written as $\Delta f = -tv_p$. Therefore, requesting an improvement in the sum of dual infeasibilities ($\Delta f > 0$) is equivalent to requesting

$$-tv_p > 0 \tag{17}$$

which can be achieved in two ways:

$$\text{If } v_p > 0 \text{ then } t < 0 \text{ must hold,} \tag{18}$$

$$\text{if } v_p < 0 \text{ then } t > 0 \text{ must hold.} \tag{19}$$

As long as there is a $v_i \neq 0$ with type$(x_{Bi}) \neq 3$ (type-3 variables are not candidates to leave the basis) there is a chance to improve the dual objective function. The precise conditions will be worked out in the sequel. From among the candidates we can select v_p using some simple or sophisticated (steepest edge type) rule.

Let k denote the original index of the pth basic variable x_{Bp}, i.e., $x_k = x_{Bp}$ (which is selected to leave the basis). At this point we stipulate that after the basis change d_k *of the outgoing variable take a feasible value.* This is not necessary but it gives a better control of dual infeasibilities.

If t moves away from zero (increasing or decreasing as needed) some of the d_js move toward zero (the boundary of their feasibility domain) either from the feasible or infeasible side and at a specific value of t they reach it. Such values of t are determined by:

$$t_j = \frac{d_j}{\alpha_j^p}, \text{ for some nonbasic } j \text{ indices}$$

and they enable a basis change since $d_j(t)$ becomes zero at this value of t, see (13). It also means that the j-th dual constraint becomes tight at this point. Let us assume the incoming variable x_q has been selected. Currently, d_k of the outgoing basic variable is zero. After the basis change its new value is determined by the transformation formula of the simplex method giving

$$\bar{d}_k = -\frac{d_q}{\alpha_{pq}} = -t_q,$$

which we want to be dual feasible. The proper sign of $\bar{d}_k$ is determined by the way the outgoing variable leaves the basis. This immediately gives rules how an incoming variable can be determined once an outgoing variable (pivot row) has been chosen. Below is a verbal description of these rules.

1. If $v_p > 0$ then $t_q < 0$ is needed for (18) which implies that the pth basic variable must leave the basis at lower bound (because $\bar{d}_k$ must be nonnegative for feasibility). In the absence of dual degeneracy this means that d_q and α_q^p must be of opposite sign. In other words, the potential pivot positions in the selected row are those that satisfy this requirement.

2. If $v_p < 0$ then $t_q > 0$ is needed which is only possible if the outgoing variable x_{Bp} (alias x_k) is of type-1 leaving at upper bound. In the absence of degeneracy this means that d_q and α_q^p must be of the same sign.

3. If $v_p \neq 0$ and the outgoing variable is of type-0 then the sign of d_q is immaterial. Therefore, to satisfy (17), if $v_p > 0$ we look for $t_q < 0$ and if $v_p < 0$ choose from the positive t values.

It remains to see how vector $\mathbf{v} = [v_1, \ldots, v_m]^T$ can be computed for row selection. In vector form, (16) can be written as

$$\mathbf{v} = \sum_{j \in \mathcal{M}} \boldsymbol{\alpha}_j - \sum_{j \in \mathcal{P}} \boldsymbol{\alpha}_j = \mathbf{B}^{-1} \left(\sum_{j \in \mathcal{M}} \mathbf{a}_j - \sum_{j \in \mathcal{P}} \mathbf{a}_j \right) = \mathbf{B}^{-1} \tilde{\mathbf{a}} \tag{20}$$

with obvious interpretation of auxiliary vector $\tilde{\mathbf{a}}$. The latter is an inexpensive operation in terms of the revised simplex method.

3.2 Analysis of the dual infeasibility function f(t)

A detailed analysis is given in [8]. Here we give the conclusions of it.

It can be investigated how the sum of dual infeasibilities, $f(t)$, changes as t moves away from 0 ($t \geq 0$ or $t \leq 0$). It can be shown that, in either case, $f(t)$ is a piecewise linear concave function with break points corresponding to different choices of the entering variable. The global maximum of this function is achieved when its slope changes sign. It gives the maximum improvement in the sum of dual infeasibilities that can be achieved with the selected outgoing variable by making multiple use of the updated pivot row.

The following cases are distinguished.

1. If $t \geq 0$ is required then the dual feasibility status of d_j (and set $\mathcal{M}$ or $\mathcal{P}$, thus the composition of $f(t)$) changes for values of t defined by positions where
   $d_j < 0$ and $\alpha_j^p < 0$ or
   $d_j \geq 0$ and $\alpha_j^p > 0$

2. If $t \leq 0$ is required then the critical values are defined by
   $d_j < 0$ and $\alpha_j^p > 0$ or
   $d_j \geq 0$ and $\alpha_j^p < 0$.

The second case can directly be obtained from the first one by using $-\alpha_j^p$ in place of α_j^p. In both cases there is a further possibility. Namely, if type$(x_j) = 3$ (free variable) and $d_j \neq 0$ then at the critical point the feasibility status of d_j changes twice (thus two ratios are defined). First when it becomes zero (feasible), and second, when it becomes nonzero again. Both cases define identical values of d_j/α_j^p for t.

Let the critical values defined above for $t \geq 0$ be arranged in an ascending order: $0 \leq t_1 \leq \cdots \leq t_Q$, where Q denotes the total number of them. For $t \leq 0$ we make a reverse ordering: $t_Q \leq \cdots \leq t_1 \leq 0$, or equivalently, $0 \leq -t_1 \leq \cdots \leq -t_Q$. Now we are ready to investigate how $f(t)$ characterizes the change of dual infeasibility.

Clearly, Q cannot be zero, i.e., if row p has been selected as a candidate it defines at least one critical value, see (16). Assuming $v_p < 0$ the initial slope of $f(t)$, according to (15), is

$$s_p^0 = -v_p = \sum_{j \in \mathcal{P}} \alpha_j^p - \sum_{j \in \mathcal{M}} \alpha_j^p. \tag{21}$$

Now $t \geq 0$ is required, so we try to move away from $t = 0$ in the positive direction. $f(t)$ keeps improving at the rate of s_p^0 until t_1. At this point $d_{j_1}(t_1) = 0$, j_1 denoting the position that defined the smallest ratio $t_1 = \dfrac{d_{j_1}(0)}{\alpha_{j_1}^p}$. At t_1 the feasibility status of d_{j_1} changes. Either it becomes feasible at this point or it becomes infeasible after t_1.

If $t_1 \geq 0$ then either (a) $d_{j_1} \geq 0$ and $\alpha_{j_1}^p > 0$ or (b) $d_{j_1} \leq 0$ and $\alpha_{j_1}^p < 0$. In these cases:

(a) $d_{j_1}(t)$ is decreasing.

- (i) If d_{j_1} was feasible it becomes infeasible and j_1 joins $\mathcal{M}$. At his point s_p^0 decreases by $\alpha_{j_1}^p$, see (21).
- (ii) If d_{j_1} was infeasible ($j_1 \in \mathcal{P}$) it becomes feasible and j_1 leaves $\mathcal{P}$. Consequently, s_p^0 decreases by $\alpha_{j_1}^p$.

If $d_{j_1} = 0$ then we only have (i).

(b) $d_{j_1}(t)$ is increasing.

- (i) If d_{j_1} was feasible it becomes infeasible and j_1 joins $\mathcal{P}$. At his point s_p^0 decreases by $-\alpha_{j_1}^p$, see (21).
- (ii) If d_{j_1} was infeasible ($j_1 \in \mathcal{M}$) it becomes feasible and j_1 leaves $\mathcal{M}$. Consequently, s_p^0 decreases by $-\alpha_{j_1}^p$.

If $d_{j_1} = 0$ then we only have (i).

Cases (a) and (b) can be summarized by saying that at t_1 the slope of $f(t)$ decreases by $|\alpha_{j_1}^p|$ giving $s_p^1 = s_p^0 - |\alpha_{j_1}^p|$. If s_p^1 is still positive we carry on with the next point (t_2), and so on. The above analysis is valid at each point. Clearly, $f(t)$ is linear between two neighboring threshold values. For obvious reasons, these values are called *breakpoints*. The distance between two points can be zero if a breakpoint has a multiplicity > 1. Since the slope decreases at breakpoints $f(t)$ *is a piecewise linear concave function* as illustrated in Figure 1. It achieves its maximum when the slope changes sign. This is a global maximum. After this point the dual objective starts deteriorating.

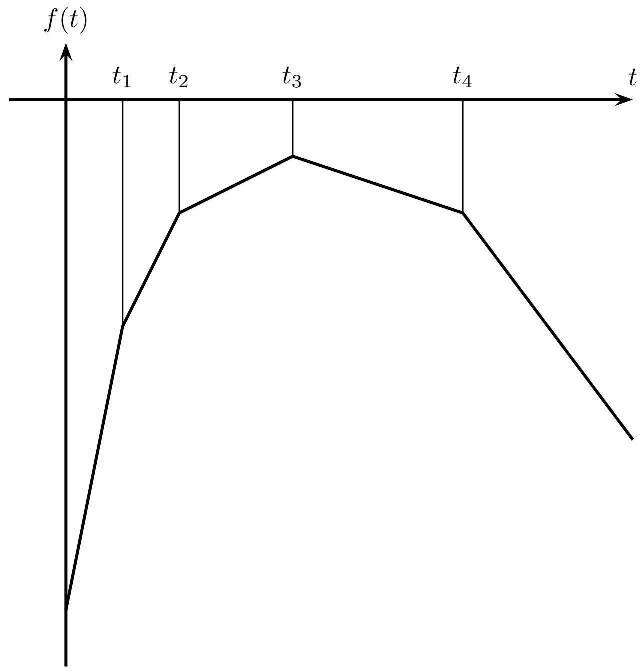

Figure 1: The sum of dual infeasibilities as a function of t.

If $v_p > 0$ then $t \leq 0$ is required. In this case the above analysis remains valid if α_j^p is substituted by $-\alpha_j^p$. It is easy to see that both cases are covered if we take $s_p^0 = |v_p|$ and

$$s_p^k = s_p^{k-1} - |\alpha_{j\,k}^p|, \text{ for } k = 1, \dots, Q.$$

3.3 A GDPO iteration step by step

Let $t_0 = 0$ and $f_k = f(t_k)$. Obviously, the sum of dual infeasibilities in the breakpoints can be computed recursively as $f_k = f_{k-1} + s_p^{k-1}(t_k - t_{k-1})$, for $k = 1, \dots, Q$.

Below, we give the description of one iteration of the algorithm called GDPO (for Generalized Dual Phase One).

An iteration of the Generalized Dual Phase-1 (GDPO) algorithm:

1. Identify sets $\mathcal{P}$ and $\mathcal{M}$ as defined in (10) and (11). If both are empty, perform feasibility correction. After that the solution is dual feasible, algorithm terminates.

2. Form auxiliary vector $\tilde{\mathbf{a}} = \sum_{j \in \mathcal{M}} \mathbf{a}_j - \sum_{j \in \mathcal{P}} \mathbf{a}_j$.

3. Compute the vector of dual phase-1 reduced costs: $\mathbf{v} = \mathbf{B}^{-1}\tilde{\mathbf{a}}$, as in (20).

4. Select an improving candidate row according to some rule (e.g., Dantzig [2] or a normalized pricing [3; 5]), denote its basic position by p. This will be the pivot row.

   If none exists, terminate: The dual problem is infeasible.

5. Compute the p-th row of $\mathbf{B}^{-1}$: $\boldsymbol{\beta}^T = \mathbf{e}_p^T\mathbf{B}^{-1}$ and determine nonbasic components of the updated pivot row by $\alpha_j^p = \boldsymbol{\beta}^T\mathbf{a}_j$ for $j \in \mathcal{R}$.

6. Compute dual ratios for eligible positions following rules discussed in section 3.2, according to $v_p < 0$, or $v_p > 0$. Store their absolute values in a sorted order: $0 \leq |t_1| \leq \cdots \leq |t_Q|$.

7. Set $k = 0,\ t_0 = 0,\ f_0 = f(0),\quad s_p^0 = |v_p|$.
   While $k < Q$ and $s_p^k \geq 0$ **do**
   $k := k + 1$
   j_k: the column index of the variable that defined the k-th smallest ratio, $|t_k|$.
   Compute $f_k = f_{k-1} + s_p^{k-1}(t_k - t_{k-1}),\ s_p^k = s_p^{k-1} - |\alpha_{j_k}^p|$.
   end while
   Let q denote the index of the last breakpoint for which the slope s_p^k was still nonnegative, $q = j_k$. The maximum of $f(t)$ is achieved at this break point. The incoming variable is x_q.

8. Compute $\boldsymbol{\alpha}_q = \mathbf{B}^{-1}\mathbf{a}_q$.

   Update basis inverse: $\bar{\mathbf{B}}^{-1} = \mathbf{E}\mathbf{B}^{-1}$, $\mathbf{E}$ denoting the elementary transformation matrix created from $\boldsymbol{\alpha}_q$ using pivot position p.

   Update the basic/nonbasic index sets.

   Update solution: Interestingly, in dual phase-1 there is no need to carry the values of the primal basic variables. They will only be needed in dual phase-2. Therefore the updating step below can be omitted which slightly speeds up the iterations. However, for completeness, the updating operations are presented below for cases when GDPO is used in conjunction with some other methods that require the updated primal basic variables.

   Update $\mathbf{x}_\mathcal{B}$ by $\bar{\mathbf{x}}_\mathcal{B} = \mathbf{E}\mathbf{x}_\mathcal{B}$, and set $\bar{x}_{Bp} = x_q + \theta_P$, where $\theta_P = x_{Bp}/\alpha_q^p$ if $v_p > 0$ or $\theta_P = (x_{Bp} - u_{Bp})/\alpha_q^p$ if $v_p < 0$.

4 Computational study of GDPO algorithm

During the theoretical analysis of the favorable features of GDPO in [8] it was often quoted that they show up more strikingly if several breakpoints are defined per iteration and the maximum of $f(t)$ is not achieved at the first one, because in this case the large flexibility of the algorithm can well be utilized. Whether it occurs in reality, one only can say if it is checked through a computational study.

In the sequel we give account of a comparative study of GDPO with the first breakpoint method which will be referred as the "traditional" dual phase-1 algorithm (TD).

4.1 Characteristics of the test problems

The purpose of the study was to investigate the effectiveness of GDPO. The test environment was the simplex based experimental code HIPLEX developed by the author. Though HIPLEX is "primal oriented" it contains most of the computational tools required by an implementation of the dual.

HIPLEX has been designed to serve as a test environment of new algorithms and algorithmic elements for the efficient solution of large scale linear programming problems. As such, it is an experimental code full of statements that gather information on the performance of the implemented algorithms. In this way it is very suitable to study the effectiveness of new elements.

In the evaluation of GDPO, *effectiveness* is defined in terms of *the number of dual phase-1 iterations.* As handling type-0 and type-1 variables in dual phase-1 is trivial (see *dual feasibility correction* [8; 9]), we have chosen problems which are dominated by type-2 (nonnegative) and type-3 (free) variables.

To make the findings reproducible only widely accepted (and accessible) problems were used. Among them there were smaller, medium and large scale ones.

Table 2 gives the main characteristics of the test problems used, namely, the number of constraints (m), number of structural variables ($\bar{n}$), number of nonzeros in $\mathbf{A}$, and the break-down of the number of structural variables by type. The problems are listed in alphabetical order of their names.

Table 2:

Problem	Rows	Columns	Nonzeros	# of variables by type			
				Type-0	Type-1	Type-2	Type-3
25fv47	822	1571	11127	0	0	1571	0
80bau3b	2262	9799	29063	498	2986	6315	0
agg	488	163	2541	0	0	163	0
agg2	516	302	4515	0	0	302	0
agg3	516	302	4531	0	0	302	0
baxter	27441	15128	109823	0	1122	14006	0
bnl1	643	1175	6129	0	0	1175	0
bnl2	2325	3489	16124	0	0	3489	0
boeing1	351	384	3865	0	228	156	0
cre_a	3516	4067	19054	0	0	4067	0
cre_b	9648	72447	328542	0	0	72447	0
cre_c	3068	3678	16922	0	0	3678	0
cre_d	8926	69980	312626	0	0	69980	0
czprob	929	3523	14173	229	0	3294	0
d6cube	415	6184	43888	0	0	6184	0
dbir2	18906	27355	1148847	0	0	27355	0
degen2	444	534	4449	0	0	534	0
degen3	1504	1818	26230	0	0	1818	0
degen4	4420	6711	107375	0	0	6711	0
grow07	140	301	2633	0	280	21	0
grow15	300	645	5665	0	600	45	0
grow22	440	946	8318	0	880	66	0
israel	174	142	2358	0	0	142	0
maros	847	1443	10006	35	0	1408	0
mod011	4481	10958	37425	1	1596	9361	0
nsct2	23003	14981	686396	0	0	14981	0
osa-07	1118	23949	167643	0	0	23949	0
osa-14	2337	52460	367220	0	0	52460	0
osa-30	4350	100024	700160	0	0	100024	0
perold	625	1376	6026	64	266	958	88
pilot_we	723	2789	9218	78	294	2337	80
rentacar	6804	9557	42019	650	179	8728	0
scagr_2	32847	34580	141757	0	0	34580	0
scrs_3	16545	17420	71401	0	0	17420	0
scsd6	147	1350	5666	0	0	1350	0
scsd8	397	2750	11334	0	0	2750	0
sctap2	1090	1880	8124	0	0	1880	0
sctap3	1480	2480	19734	0	0	2480	0
ship08l	778	4283	17085	0	0	4283	0
ship12l	1151	5427	21597	0	0	5427	0
stair	357	467	3857	82	6	373	6
sto27	14441	34114	114973	0	0	34114	0
stocfor2	2157	2031	9492	0	0	2031	0
stocfor3	16676	15695	74004	0	0	15695	0
sws	14310	12465	105480	0	0	12465	0
unicolns	5421	45569	168220	2	1449	44118	0
wood1p	244	2594	70216	0	0	2594	0
woodw	1098	8405	37478	0	0	8405	0

4.2 Evaluation of the test runs

First, the results are shown in a "raw" tabular form in Table 3 followed by further tables that reflect the conclusions obtained from the raw version.

Table 3 shows the number of dual phase-1 iterations required by GDPO and TD, respectively. The solution strategy for each comparative run is also included to identify scaling, the selection of starting basis (logical or crash [11]), and whether Devex pricing was used.

Particularly interesting are the columns giving the ratio of the iteration counts. T/G means how many times more iterations were made with the traditional method than with GDPO. Column G/T is the reciprocal of it.

It can be seen that run strategies were not identical for all problems. There are several reasons for that. First, we had to choose dual infeasible starting bases. When the all-logical did not satisfy this a crash basis was used. Second, in case of larger problems dual Devex was used to obtain more reasonable run times. Third, to avoid numerical difficulties, several problems were scaled prior to solution. Presolve was not applied to any of the problems. It is important to note that for any given problem both GDPO and TD was run with identical settings which is included in the table.

The basis of the expected effective operation of GDPO is the multiple use of the updated pivot row which is measured by the number of breakpoints used for the maximization of $f(t)$. In some sense we can view this measure as an *algorithmic steplength*. Table 4 demonstrates this feature of GDPO. As there is a huge variation in the figures some aggregation was necessary to be able display the findings. Any row in the table shows how many times was the first, second, ..., 5th breakpoint the maximizer of the $f(t)$ of an iteration, how many times was the maximizing breakpoint in the intervals $6-10$, $11-20$, $21-50$ and how many times were used more than 50 breakpoints (50+). The total number of phase-1 iterations is shown in the last column. The aggregate part of the table hides many interesting cases, in particular the 50+ column. To somewhat relieve this problem we introduced a column headed by "Max" which shows the maximum number of breakpoints used in one iteration. For instance, in the row of `mod011` we can see that from the 602 iterations in phase-1 (last column) it happened 31 times that the maximum of $f(t)$ was achieved at the 5th breakpoint. Furthermore, there was an iteration (Max) when 917 breakpoints were needed to obtain the maximum of $f(t)$.

The starting point for the assessment of the effectiveness of GDPO is Table 3. It can be seen that GDPO is more effective than TD in all but three cases. Column T/G shows how many times more iterations were needed by TD in dual phase-1. Entries greater than 1 show cases when GDPO was better. At the same time, this number can also be viewed as the measure of effectiveness. In three cases the number was slightly smaller than 1 indicating that in these cases TD was slightly more effective. For a better overview a summary table 5 is provided to show in how many cases and how many times was GDPO more effective than TD.

Table 3:

Problem	Initial # of dual infeasibilities	# of dual ph-1 itns GDPO	TD ($k=1$)	Ratios T/G	G/T	Solution strategy Scaling Y/N	Start bas. CB/LB	Devex Y/N
25fv47	41	238	1033	4.34	0.23	Y	LB	N
80bau3b	208	736	997	1.35	0.74	Y	LB	Y
agg	95	11	107	9.73	0.10	Y	LB	N
agg2	171	13	109	8.38	0.12	Y	LB	N
agg3	171	13	109	8.38	0.12	Y	LB	N
baxter	3259	2687	4482	1.67	0.60	Y	CB	Y
bnl1	57	4	27	6.75	0.15	Y	LB	Y
bnl2	156	49	134	2.73	0.36	Y	LB	Y
boeing1	164	9	102	11.33	0.09	Y	LB	N
cre_a	1156	476	1655	3.48	0.29	N	CB	Y
cre_b	14503	4604	12281	2.67	0.37	N	CB	Y
cre_c	1056	532	1559	2.93	0.34	N	CB	Y
cre_d	10409	3479	14798	4.25	0.23	N	CB	Y
czprob	1521	191	1959	10.26	0.10	Y	LB	N
d6cube	2637	1209	6500	5.38	0.19	Y	CB	Y
dbir2	9210	9631	9848	1.02	0.98	Y	LB	Y
degen2	425	143	574	4.01	0.25	Y	LB	Y
degen3	1249	571	1945	3.41	0.29	Y	LB	Y
degen4	2697	1177	6792	5.77	0.17	Y	LB	Y
grow07	21	1	14	14.00	0.07	Y	CB	N
grow15	45	1	14	14.00	0.07	Y	CB	N
grow22	66	1	14	14.00	0.07	Y	CB	N
israel	24	1	24	24.00	0.04	Y	LB	N
maros	162	666	799	1.20	0.83	Y	LB	N
mod011	4343	602	3753	6.23	0.16	Y	CB	Y
nsct2	11240	11507	11636	1.01	0.99	Y	LB	Y
osa-07	9201	114	3456	30.32	0.03	Y	CB	Y
osa-14	19695	141	3616	25.65	0.04	Y	CB	Y
osa-30	37495	68	7785	114.49	0.01	Y	CB	Y
perold	7	725	587	0.81	1.24	Y	LB	N
pilot_we	91	580	1065	1.84	0.54	Y	LB	N
rentacar	2	1778	1629	0.92	1.09	Y	LB	N
scagr_2	8645	16676	27473	1.65	0.61	Y	LB	Y
scrs_3	4355	11635	12765	1.10	0.91	Y	LB	Y
scsd6	218	46	147	3.20	0.31	Y	CB	N
scsd8	353	6	30	5.00	0.20	Y	CB	N
sctap2	238	442	578	1.31	0.76	Y	CB	Y
sctap3	315	532	714	1.34	0.75	Y	CB	Y
ship08l	581	39	587	15.05	0.07	Y	CB	N
ship12l	708	51	958	18.78	0.05	Y	CB	N
stair	1	180	152	0.84	1.18	Y	LB	N
sto27	11541	4879	6844	1.40	0.71	Y	CB	Y
stocfor2	639	1366	1552	1.14	0.88	Y	LB	N
stocfor3	5077	10620	11848	1.12	0.90	Y	LB	N
sws	2190	988	1694	1.71	0.58	Y	CB	Y
unicolns	43914	4975	50841	10.22	0.10	Y	CB	Y
wood1p	1057	30	1288	42.93	0.02	Y	CB	N
woodw	1738	60	2520	42.00	0.02	Y	CB	N

Table 4:

Problem	Number of breakpoints used 1	2	3	4	5	6–10	11–20	21–50	50+	Max	# of itns in ph-1
25fv47	80	76	38	16	7	17	4	–	–	18	238
80bau3b	389	165	66	41	29	41	5	–	–	14	736
agg	–	1	–	2	–	6	2	–	–	18	11
agg2	1	1	–	3	1	4	–	3	–	48	13
agg3	1	1	–	3	1	4	–	3	–	48	13
baxter	1381	331	157	142	74	201	188	178	35	194	2687
bnl1	–	–	–	–	–	–	4	–	–	17	4
bnl2	17	18	4	–	–	–	10	–	–	19	49
boeing1	1	–	–	–	2	3	2	–	1	136	9
cre_a	79	62	39	27	30	67	79	57	36	474	476
cre_b	103	179	169	203	171	702	287	875	1915	3538	4604
cre_c	102	93	62	49	20	89	49	47	21	397	532
cre_d	108	105	151	109	133	548	656	733	936	3948	3479
czprob	46	71	32	12	8	5	2	4	11	172	191
d6cube	103	90	83	81	92	300	231	139	90	821	1209
dbir2	8851	587	130	34	22	6	1	–	–	13	9631
degen2	19	34	64	7	5	11	–	1	2	58	143
degen3	128	131	247	23	11	14	11	3	3	91	571
degen4	312	165	189	109	111	185	90	10	6	162	1177
grow07	–	–	–	–	–	–	–	1	–	21	1
grow15	–	–	–	–	–	–	–	1	–	45	1
grow22	–	–	–	–	–	–	–	–	1	66	1
israel	–	–	–	–	–	–	–	1	–	24	1
maros	258	200	97	44	24	34	8	1	–	32	666
mod011	260	107	64	46	31	52	17	10	15	917	602
nsct2	10974	298	85	35	19	59	31	6	–	26	11507
osa-07	37	24	3	6	1	2	4	5	32	4503	114
osa-14	70	18	4	2	1	2	3	6	39	9454	145
osa-30	18	3	4	4	2	2	1	6	39	3654	79
perold	414	156	47	18	21	38	15	14	2	154	725
pilot_we	341	84	39	16	11	42	36	9	2	103	580
rentacar	1737	33	6	–	–	1	1	–	–	11	1778
scagr_2	10625	5186	–	865	–	–	–	–	–	4	16676
scrs_3	8311	2995	298	23	8	–	–	–	–	5	11635
scsd6	2	2	4	2	2	6	14	10	4	100	46
scsd8	–	–	–	1	–	1	–	1	3	260	6
sctap2	65	98	58	64	26	83	28	17	3	55	442
sctap3	103	124	46	59	39	87	39	32	3	90	532
ship08	16	2	–	1	12	–	–	–	8	73	39
ship12	13	6	9	9	1	1	–	–	12	62	51
stair	147	30	1	–	1	–	1	–	–	14	180
sto27	612	725	916	520	484	1006	424	191	1	56	4879
stocfor2	583	450	179	99	30	25	–	–	–	10	1366
stocfor3	3730	3491	1604	761	443	546	44	1	–	21	10620
sws	367	332	12	85	31	71	70	–	–	17	988
unicolns	441	627	225	359	96	348	2659	220	–	34	4975
wood1p	–	–	–	–	–	–	5	8	17	588	30
woodw	–	2	–	2	4	7	7	15	23	801	60

In general, a 25% improvement of an optimization algorithm is viewed remarkable. If we raise it to 50% than it can be seen that GDPO achieves this improvement in 35 cases out of the total of 48, see Table 5. In particular, in 13 cases the effectiveness improved more than 10 times. The performance of GDPO on the `osa` family of problems proved to be quite outstanding where, in the best case (`osa-30`), the improvement was 114×.

Algorithms that use the first breakpoint (like TD) can reduce the number of dual infeasibilities only one by one (except when degeneracy helps achieve more). Though GDPO is monotone only in the sum of infeasibilities it is able to reduce the number of infeasibilities in one iteration quite dramatically. The best examples of this situation are shown in Table 6 (altogether 20 problems).

Table 4 demonstrates that GDPO actively uses the breakpoints of $f(t)$. Even more can be seen. The theoretically best case is to eliminate all dual infeasibilities in a single iteration. This table shows that this best performance is actually achieved on real life problems. They are the `grow` family (`grow07`, `grow15`, `grow22`), and `israel`. In the `grow` problems there are relatively few type-2 variables (21, 45 and 66, resp.). However, if we start with a crash basis all dual logicals corresponding to these positions are dual infeasible. The $f(t)$ function defined in the first iteration of these problems achieves its maximum by using up all breakpoints (21, 45 and 66 [the same as the number of type-2 variables]) and it makes all dual logicals feasible in one iteration. In `israel` all variables are type-2 but GDPO was able to achieve the theoretically best possible effectiveness even in this case.

Table 5: Efficiency of GDPO measured in the number of iterations

Improvement	Number of times
$1.0 - 1.5\times$	10
$1.6 - 3.0\times$	7
$3.1 - 5.0\times$	7
$5.1 - 10.0\times$	8
More than $10\times$	13
Deterioration	
$0.8 - 1.0\times$	3
Total	48

Table 6: Particularly fast reduction of the number of dual infeasibilities to achieving dual feasibility

Problem	Initial # of dual inf.	GDPO iterations
agg	95	11
agg2	171	13
agg3	171	13
bnl1	57	4
boeing1	164	9
grow07	21	1
grow15	45	1
grow22	66	1
israel	24	1
mod011	4343	602
osa-07	9201	114
osa-14	19695	141
osa-30	37495	68
scsd6	218	46
scsd8	353	6
ship08l	581	39
ship12l	708	51
unicolns	43914	4975
wood1p	1057	30
woodw	1738	60

5 Conclusions

The purpose of this paper was to study the computational performance of the GDPO dual phase-1 algorithm [8; 9].

Experience obtained through the theoretical and computational investigations of GDPO can be interpreted and summarized as follows.

1. GDPO contains the "first breakpoint" algorithms as special cases thus it is a generalization of them.

2. GDPO is capable of multiply utilizing the updated pivot row and thus making a progress that corresponds to several traditional iterations.

3. GDPO is monotone only in the sum of infeasibilities which opens up a huge flexibility enabling the choice of a properly sized pivot which results in substantially better numerical characteristics.

4. In case of dual degeneracy GDPO has a much better chance to make a non-degenerate iteration.

5. GDPO can be implemented easily and the iteration speed hardly deteriorates compared to TD if some advanced techniques of computer science are used.

6. The theoretically favorable features of GDPO do materialize in practice to a large extent.

7. Regarding effectiveness, GDPO supersedes the traditional "first breakpoint" method nearly always. In several real problems it can work with maximum effectiveness, i.e., can make the solution dual feasible in one non-trivial iteration.

8. The main reason for the favorable performance of GDPO is that it makes the maximum progress towards dual feasibility that can be achieved with a given outgoing variable which otherwise would be possible only by many traditional dual iterations. If many breakpoints are used the difference can be very substantial.

Based on the above we can conclude that GDPO is both theoretically and computationally an important algorithm that is well positioned to be included in the toolbox of modern simplex implementations.

References

[1] G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T.C. Koopmans, editor, *Activity analysis of production and allocation*, pages 339–347. Wiley, New York, 1951.

[2] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, 1963.

[3] J.J.H. Forrest and D. Goldfarb. Steepest edge simplex algorithms for linear programming. *Mathematical Programming*, 57(3):341–374, 1992.

[4] R. Fourer. Notes on the dual simplex method. Unpublished, March 1994.

[5] P.M.J. Harris. Pivot Selection Method of the Devex LP Code. *Mathematical Programming*, 5:1–28, 1973.

[6] C.E. Lemke. The Dual Method of Solving the Linear Programming Problem. *Naval Research Logistics Quarterly*, 1:36–47, 1954.

[7] I. Maros. A Piecewise Linear Dual Procedure in Mixed Integer Programming. In F. Giannesi, S. Komlósi, and T. Rapcsák, editors, *New Trends in Mathematical Programming*, pages 159–170. Kluwer Academic Publishers, 1998.

[8] I. Maros. A Piecewise Linear Dual Phase-1 Algorithm for the Simplex Method. *Computational Optimization and Applications*, 26:63–81, 2003.

[9] I. Maros. *Computational Techniques of the Simplex Method*, volume 61 of *International Series in Operations Research and Management*. Kluwer Academic Publishers, Boston, 2003. 325+xx pages, Research monograph.

[10] I. Maros and G. Mitra. Simplex Algorithms. In J. Beasley, editor, *Advances in Linear and Integer Programming*, pages 1–46. Oxford University Press, 1996.

[11] I. Maros and G. Mitra. Strategies for Creating Advanced Bases for Large-Scale Linear Programming Problems. *INFORMS Journal on Computing*, 10(2):248–260, Spring 1998.