An Enhanced Piecewise Linear Dual Phase-1 Algorithm for the Simplex Method

István Maros

Department of Computing, Imperial College, London

Email: i.maros@ic.ac.uk

Departmental Technical Report 2002/15*

ISSN 1469–4174

Abstract

A dual phase-1 algorithm for the simplex method that handles all types of variables is presented. In each iteration it maximizes a piecewise linear function of dual infeasibilities in order to make the largest possible step towards dual feasibility with a selected outgoing variable. The algorithm can be viewed as a generalization of traditional phase-1 procedures. It is based on the multiple use of the expensively computed pivot row. By small amount of extra work per iteration, the progress it can make is equivalent to many iterations of the traditional method. While this is its most important feature, it possesses some additional favorable properties, namely, it can be efficient in coping with degeneracy and numerical difficulties. Both theoretical and computational issues are addressed. Some computational experience is also reported which shows that the potentials of the method can materialize on real world problems. This paper is based on IC Departmental Technical Report 2000/13 and contains an enhancement of the main algorithm.

*Updated: 2004.01.11

Keywords: Linear programming, Dual simplex method, Phase-1, Piecewise linear functions.

1 Introduction

The dual simplex algorithm (DSA) developed by Lemke [8] has long been known as a better alternative to Dantzig's primal simplex [2] for solving certain types of linear programming (LP) problems, mostly in cases where a dual feasible solution is readily available. This latter situation is typical in the LP relaxation of mixed integer programming problems within branch and bound type solution algorithms. In such a case the optimal basic solution to a node problem is dual feasible for the immediate successor problems. However, in many other cases a dual feasible solution is not available and the dual algorithm cannot be used even if it would be advantageous.

There are some techniques to obtain a dual basic feasible solution for the problem. The most typical one is the dual variant of the big-M method. In this case an extra '$\leq$' type constraint on $\sum x_j$ is added to the original constraints. It is made non-binding by setting the right-hand-side coefficient equal to a large number (the big M). This enables the gradual build-up of a dual feasible basis if one exists. For details c.f., [13]. This and the other methods have been worked out for the case when the LP problem is in the standard form. If all types of variables are present in a problem the situation is more complicated. In such a case, one possibility is to introduce additional variables and constraints to convert the problem into the standard form with nonnegative variables only and apply the big-M or other methods.

In this paper we propose an algorithm to obtain a dual feasible solution for LP problems with all types of variables. Our motivation was the computational enhancement of dual phase-1 in this general case. The key idea of this approach is the multiple use of the (expensively) updated pivot row. The algorithm is a modification of the DSA such that in each iteration the longest possible step is made with a selected outgoing variable towards dual feasibility. This is achieved by maximizing a concave piecewise linear function in every iteration. In this sense, it is a greedy algorithm. We show that the extra work per iteration is little. At the same time, the algorithm can lead to a considerable enhancement of efficiency by greatly reducing the number of iterations in dual phase-1 though there is no theoretical guarantee for that. This algorithm is monotone in the sum of dual infeasibilities. As such, the number of infeasibilities can even increase (though in this case the sum will definitely decrease), remain the same (even in the nondegenerate case, though infeasible positions may change) or decrease. The algorithm has some inherent flexibility that can alleviate or overcome occasional numerical and/or algorithmic (like degeneracy) difficulties.

The rest of the paper is organized as follows. In section 2 we state the primal and dual problems with all types of variables and discuss relevant work done in this area. Section 3 gives an analysis of dual infeasibility, it introduces a dual phase-1 procedure and gives its algorithmic description. Section 4 presents a numerical examples that illustrates how the procedure works in practice. In section 5 the results of a limited computational testing are presented and discussed. It is followed by a summary and conclusions in section 6.

2 Problem statement

2.1 The primal problem

Consider the following primal linear programming (LP) problem:

$$\begin{array}{ll} \text{minimize} & c^T x, \\ \text{subject to} & Ax = b, \\ & l \leq x \leq u, \end{array}$$

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

By some elementary transformations it can be achieved that the variables (whether logical or structural) fall into four categories as shown in Table 1 (for further details, see Orchard-Hays [12]).

Table 1: Types of variables

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

2.2 The dual problem

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

$$\begin{array}{lll} (P1) & \text{minimize} & c^T x, \\ & \text{subject to} & Ax = b, \\ & & 0 \leq x \leq u, \end{array}$$

where all components of u are finite.

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

$$x_B = B^{-1}\left(b - \sum_{j \in U} u_j a_j\right),$$

where 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 $d_j = c_j - \pi^T a_j = c_j - c_B^T B^{-1} a_j$ which is further equal to $c_j - c_B^T \alpha_j$ if the notation $\alpha_j = B^{-1} a_j$ is used.

The dual of (P1) is:

$$\begin{array}{lll} (D1) & \text{maximize} & b^T y - u^T w, \\ & \text{subject to} & A^T y - w \leq c, \\ & & w \geq 0, \end{array}$$

where $y \in \mathbb{R}^m$ and $w \in \mathbb{R}^n$ are the dual variables. 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 [12]. 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 upper bounded version of the DSA was first described by Orchard-Hays [12] and later by Chvátal [1]. They both deal with dual phase-2. Maros published a general dual phase-2 algorithm (BSD in [10]) that handles all types of variables and has some favorable computational properties.

Phase-1 procedures, whether primal or dual, compute ratios to determine the pivot element that ultimately defines the basis change. With different choices of the pivot the change in the sum and number of infeasibilities can be controlled. The classical Dantzig ratio test chooses the smallest of the ratios. It guarantees that, in case of a nondegenerate iteration the sum of infeasibilities decreases and the number of infeasibilities does not increase. Other choices can ensure that (a) either the sum of infeasibilities decreases and the number of infeasibilities stays the same, or the number of infeasibilities decreases, or (b) the sum of infeasibilities decreases.

For phase-1 of the primal with all types of variables there exist algorithms that can make long steps with an incoming variable (Wolfe [15], Greenberg [6], Maros [9]). They are all based on piecewise linear minimization/maximinaztion. The natural extension of the idea to the dual has not been investigated extensively. In fact, for dual phase-1 the only relevant work appears to be [4] in which Fourer gives a theoretical discussion of a procedure that is based on the above advanced ideas for the primal. His algorithm falls into category (a). The algorithm to be presented in this paper, which we call *GDPO (Generalized Dual Phase One)*, is monotone only in the sum of infeasibilities (category(b)).

The creation of the updated pivot row p, i.e., the computation of α_{pj} for all nonbasic indices j is an expensive operation in SSX (c.f. [11]). Traditional dual methods based on the Dantzig type pivot selection make *one iteration* with the pivot row and discard it. GDPO makes *one step* with the pivot row but it 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. As a result, GDPO seems to be worth for consideration as a general dual phase-1 algorithm.

3 Dual phase-1 with all types of variables

If there are only type-2 variables in the problem then w is not present in (D1). In this case, assuming that B is a basis to A, dual feasibility is expressed by

$$\begin{aligned} &B^T y = c_B \\ &N^T y \leq c_N, \quad \text{or } d_N = c_N - N^T y \geq 0. \end{aligned}$$

In practice, of course, all types of variables are present in a problem and it is desirable to handle them algorithmically rather than introducing additional variables and constraints and reverting to the traditional formulation. The d_j of a basic variable is zero. For nonbasic variables the dual feasible values are shown in Table 2 (for proof c.f. [12]).

Since the d_j of a type-0 variable is always feasible such variables can be, and in fact are, ignored in dual phase-1. Any d_j that falls outside the feasibility range is dual infeasible.

Table 2: Dual feasibility of nonbasic d_js (primal minimization)

Type of nonbasic variable	Dual feasibility
0	d_j of any sign
1	$d_j \geq 0$ if $x_j = 0$ $d_j \leq 0$ if $x_j = u_j$
2	$d_j \geq 0$
3	$d_j = 0$

We define two index sets of dual infeasible variables:

$$M = \{j : (x_j = 0 \text{ and } d_j < 0)\},$$

and

$$P = \{j : (x_j = u_j \text{ and } d_j > 0) \text{ or } (\text{type}(x_j) = 3 \text{ and } d_j > 0)\},$$

where type(x_j) denotes the type of x_j.

There is an easy way to make the d_j of upper bounded variables feasible. They can be infeasible in two different ways. Accordingly, we define two index sets:

$$T^+ = \{j : \text{ type}(x_j) = 1 \text{ and } j \in P\}$$
$$T^- = \{j : \text{ type}(x_j) = 1 \text{ and } j \in M\}$$

If we perform a bound flip for all such variables, the corresponding d_js become feasible. In this case the basis remains unchanged but the solution has to be updated:

$$x_B := x_B - \sum_{j \in T^+} u_j \alpha_j + \sum_{j \in T^-} u_j \alpha_j, \tag{1}$$

where $\alpha_j = B^{-1} a_j$, ':=' denotes assignment, and the sum is defined to be 0 if the corresponding index set is empty. Computing α_j for all j in (1) would be relatively expensive.

However, this entire operation can be performed in one single step for all variables involved in the following way.

$$\begin{aligned} x_B &:= x_B - \sum_{j\in T^+} u_j\alpha_j + \sum_{j\in T^-} u_j\alpha_j \\ &= x_B - B^{-1}\left(\sum_{j\in T^+} u_j a_j - \sum_{j\in T^-} u_j a_j\right) \\ &= x_B - B^{-1}\tilde{a} \end{aligned} \tag{2}$$

with the obvious interpretation of $\tilde{a}$. Having constructed $\tilde{a}$, we need only one FTRAN operation with the inverse of the basis.

Assuming that such a *dual feasibility correction* has been carried out the definition of P simplifies to:

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

While this redefinition is not really necessary at this stage it will be useful for the new algorithm.

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

The straightforward way of making the dual logicals feasible suggests that their feasibility can be disregarded during dual phase-1. As type-0 variables do not play any role at all, only type-2 and -3 variables need to be considered. If the corresponding dual logicals all have become feasible a single feasibility correction can make the logicals of the type-1 variables feasible.

As a consequence, set M is redefined to be

$$M = \{j : d_j < 0 \text{ and type}(x_j) \geq 2\}. \tag{4}$$

Using infeasibility sets of (3) and (4), the sum of dual infeasibilities can be defined as

$$f = \sum_{j\in M} d_j - \sum_{j\in P} d_j, \tag{5}$$

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.

The dual simplex method performs basis changes using the computational tools of the primal. However, in the dual the pivot row (the outgoing variable) is selected first, followed by a dual ratio test to determine the pivot column (incoming variable).

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 d_N. If this multiplier is denoted by t then the transformed value of each d_j can be written as a function of t:

$$d_j^{(p)}(t) = d_j - t\alpha_{pj}. \tag{6}$$

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

$$f^{(p)}(t) = \sum_{j\in M} d_j^{(p)}(t) - \sum_{j\in P} d_j^{(p)}(t) = f^{(p)}(0) - t\left(\sum_{j\in M}\alpha_{pj} - \sum_{j\in P}\alpha_{pj}\right).$$

Clearly, f of (5) can be obtained as $f = f^{(p)}(0)$. To simplify notations, we drop the superscript from both $d_j^{(p)}(t)$ and $f^{(p)}(t)$ and will use $d_j(t)$ and $f(t)$ instead.

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

$$\Delta f = f(t) - f(0) = -t\left(\sum_{j\in M}\alpha_{pj} - \sum_{j\in P}\alpha_{pj}\right). \tag{7}$$

Introducing notation

$$v_p = \sum_{j\in M}\alpha_{pj} - \sum_{j\in P}\alpha_{pj} \tag{8}$$

(7) 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

$$-tv_p > 0 \tag{9}$$

which can be achieved in two ways:

$$\text{If } v_p > 0 \text{ then } t < 0 \text{ must hold,} \tag{10}$$

$$\text{if } v_p < 0 \text{ then } t > 0 \text{ must hold.} \tag{11}$$

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:

$$t_j = \frac{d_j}{\alpha_{pj}}, \text{ for some nonbasic } j \text{ indices}$$

and they enable a basis change since $d_j(t)$ becomes zero at this value of t, see (6). 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. Details are given in the next section.

1. If $v_p > 0$ then $t_q < 0$ is needed for (10) 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_{pq} 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_{pq} 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 (9), 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 $v = [v_1, \ldots, v_m]^T$ can be computed for row selection. In vector form, (8) can be written as

$$v = \sum_{j \in M} \alpha_j - \sum_{j \in P} \alpha_j = B^{-1} \left(\sum_{j \in M} a_j - \sum_{j \in P} a_j \right) = B^{-1} \tilde{a} \tag{12}$$

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

3.1 Analysis of the dual infeasibility function

We can investigate how the sum of dual infeasibilities, $f(t)$, changes as t moves away from 0 ($t \geq 0$ or $t \leq 0$). We show 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.

Let the index of the pivot row be denoted by p, the outgoing variable by x_{Bp} ($\equiv x_k$) and the index of the incoming variable by q. The pivot element is α_{pq}.

After the basis change, the new values of d_j are determined by:

$$\bar{d}_j = d_j - \frac{d_q}{\alpha_{pq}} \alpha_{pj} \quad \text{for} \quad j \in N, \tag{13}$$

and for the leaving variable:

$$\bar{d_k} = -\frac{d_q}{\alpha_{pq}}.$$

The feasibility status of a d_j (described in Table 2) may change as t moves away from zero. The following analysis uses (6), (9) and (13) to keep track of the changes of the feasibility status of each $d_j(t)$.

I. $t \geq 0$, i.e., the outgoing variable leaves at upper bound.

(a) $\alpha_{pj} > 0$, $d_j(t)$ is decreasing

i. $d_j(0) > 0$

A. If $d_j(0)$ is infeasible, i.e., $j \in P$, it remains so as long as $t < \dfrac{d_j(0)}{\alpha_{pj}}$ and it becomes infeasible again if $t > \dfrac{d_j(0)}{\alpha_{pj}}$ when j joins M.

B. If $d_j(0)$ is feasible, $d_j(t)$ remains feasible as long as $t \leq \dfrac{d_j(0)}{\alpha_{pj}}$ after which it becomes negative and j joins M.

ii. If $d_j(0) = 0$ it remains feasible only if $t = \dfrac{d_j(0)}{\alpha_{pj}} = 0$. For $t > 0$ it becomes infeasible and j joins M.

(b) $\alpha_{pj} < 0$, $d_j(t)$ is increasing

i. If $d_j(0) < 0$, i.e., $j \in M$, $d_j(t)$ remains infeasible as long as $t < \dfrac{d_j(0)}{\alpha_{pj}}$. If type $(x_j) = 3$, it becomes infeasible again when $t > \dfrac{d_j(0)}{\alpha_{pj}}$ and j joins P.

ii. If $d_j(0) = 0$ and type$(x_j) = 3$ then it remains feasible only for $t = \dfrac{d_j(0)}{\alpha_{pj}} = 0$; for $t > 0$ it becomes positive and j joins P.

II. $t \leq 0$, i.e., the outgoing variable leaves at zero.

(a) $\alpha_{pj} > 0$, $d_j(t)$ is increasing

i. If $d_j(0) < 0$, i.e., $j \in M$, $d_j(t)$ remains infeasible as long as $t > \dfrac{d_j(0)}{\alpha_{pj}}$. If type$(x_j) = 3$, it becomes infeasible again when $t < \dfrac{d_j(0)}{\alpha_{pj}}$ and j joins P.

   ii. If $d_j(0) = 0$ and type$(x_j) = 3$ then $d_j(t)$ remains feasible only for $t = \dfrac{d_j(0)}{\alpha_{pj}} = 0$; for $t < 0$ it becomes positive and j joins P.

(b) $\alpha_{pj} < 0$, $d_j(t)$ is decreasing

   i. $d_j(0) > 0$

      A. If $d_j(0)$ is infeasible, i.e., $j \in P$, it remains so as long as $t > \dfrac{d_j(0)}{\alpha_{pj}}$ and it becomes infeasible again if $t < \dfrac{d_j(0)}{\alpha_{pj}}$ when j joins M.

      B. If $d_j(0)$ is feasible, $d_j(t)$ remains feasible as long as $t \geq \dfrac{d_j(0)}{\alpha_{pj}}$ after which it becomes negative and j joins M.

   ii. If $d_j(0) = 0$ it remains feasible only if $t = \dfrac{d_j(0)}{\alpha_{pj}} = 0$. For $t < 0$ it becomes infeasible and j joins M.

The above discussion can be summarized as follows.

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

2. If $t \leq 0$ is required then the critical values are defined by
   $d_j < 0$ and $\alpha_{pj} > 0$ or
   $d_j \geq 0$ and $\alpha_{pj} < 0$.

The second case can directly be obtained from the first one by using $-\alpha_{pj}$ in place of α_{pj}. 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_{pj} 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 (8). Assuming $v_p < 0$ the initial slope of $f(t)$, according to (7), is

$$s_p^0 = -v_p = \sum_{j \in P} \alpha_{pj} - \sum_{j \in M} \alpha_{pj}. \tag{14}$$

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 $t_1 = \dfrac{d_{j_1}(0)}{\alpha_{pj_1}}$. 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 $\alpha_{pj_1} > 0$ or (b) $d_{j_1} \leq 0$ and $\alpha_{pj_1} < 0$. In these cases:

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

  (i) If d_{j_1} was feasible it becomes infeasible and j_1 joins M. At his point s_p^0 decreases by α_{pj_1}, see (14).

  (ii) If d_{j_1} was infeasible ($j_1 \in P$) it becomes feasible and j_1 leaves P. Consequently, s_p^0 decreases by α_{pj_1}.

  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 P. At his point s_p^0 decreases by $-\alpha_{pj_1}$, see (14).

  (ii) If d_{j_1} was infeasible ($j_1 \in M$) it becomes feasible and j_1 leaves M. Consequently, s_p^0 decreases by $-\alpha_{pj_1}$.

  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 $|\alpha_{pj_1}|$ giving $s_p^1 = s_p^0 - |\alpha_{pj_1}|$. 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.

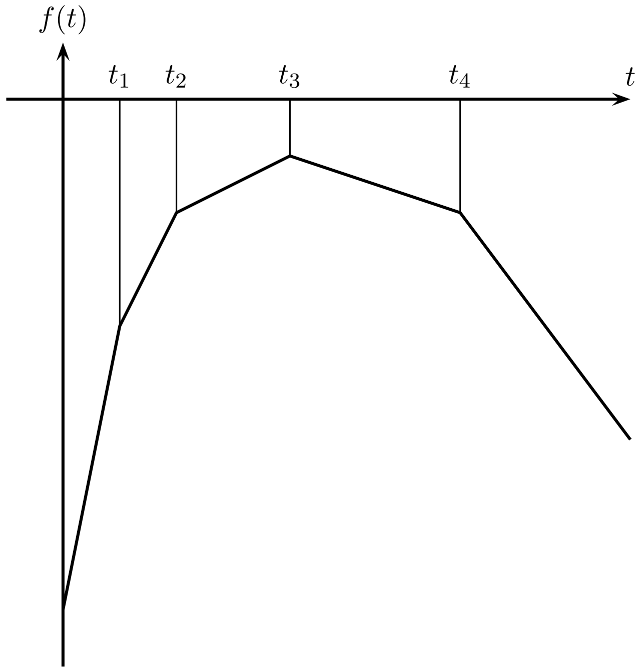

Figure 1: The sum of dual infeasibilities as a function of t.

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

$$s_p^k = s_p^{k-1} - |\alpha_{pj_k}|, \text{ for } k = 1, \ldots, Q.$$

3.2 A dual phase-1 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 which we call GDPO (for Generalized Dual Phase One).

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

1. Identify sets P and M as defined in (3) and (4). If both are empty, perform feasibility correction. After that the solution is dual feasible, algorithm terminates.

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

3. Compute the vector of dual phase-1 reduced costs: $v = B^{-1}\tilde{a}$, as in (12).

4. Select an improving candidate row according to some rule (e.g., Dantzig [2] or a normalized pricing [3, 7]), 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 B^{-1}: $\beta^T = e_p^T B^{-1}$ and determine nonbasic components of the updated pivot row by $\alpha_{pj} = \beta^T a_j$ for $j \in N$.

6. Compute dual ratios for eligible positions according to rules under **I.**, if $v_p < 0$, or **II.**, if $v_p > 0$, as discussed in section 3.1. Store their absolute values in a sorted order: $0 \le |t_1| \le \cdots \le |t_Q|$.

7. Set $k = 0,\ t_0 = 0,\ f_0 = f(0),\ \ s_p^0 = |v_p|$.

```
**While** $k < Q$ and $s_p^k \ge 0$ **do**
$k := k + 1$
$j_k$: the column index of the variable that defined the $k$-th smallest ratio, $|t_k|$.
Compute $f_k = f_{k-1} + s_p^{k-1}(t_k - t_{k-1}),\ s_p^k = s_p^{k-1} - |\alpha_{pj_k}|$.
**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 $\alpha_q = B^{-1}a_q$.

   Update basis inverse: $\bar{B}^{-1} = EB^{-1}$, E denoting the elementary transformation matrix created from α_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 x_B by $\bar{x}_B = Ex_B$, and set $\bar{x}_{Bp} = x_q + \theta_P$, where $\theta_P = x_{Bp}/\alpha_{pq}$ if $v_p > 0$ or $\theta_P = (x_{Bp} - u_{Bp})/\alpha_{pq}$ if $v_p < 0$.

3.2.1 Correctness of the algorithm

It remains to see that GDPO is a correct algorithm.

First, as the dual objective function defined in (5) has an upper bound of 0, the solution is never unbounded.

Second, maximizing f subject to the dual feasibility constraints is a convex problem. It entails that if there is no locally improving direction from a point with respect to the current objective function (sum of infeasibilities) then the same is true globally. Therefore, if no improving row can be found and the dual solution is still infeasible then the problem is dual infeasible.

Third, if there is always a positive step towards feasibility no basis can be repeated as different bases have different infeasibility values which ensures finiteness. If degeneracy is present and GDPO can make only a degenerate step then the theoretically safe (and computationally efficient) ‘ad hoc’ method by Wolfe [14] can be used as long as necessary.

3.2.2 Work per iteration

The bulk of the computational effort of an iteration of the dual is spent in the following operations.

Computing the elements of the updated pivot row (Step 5 of GDPO) requires the extraction of row p of B^{-1} which is achieved by solving $B^T\beta = e_p$ for β.

The updated form of the incoming column (part of Step 8 of GDPO) requires the solution of $B\alpha_q = a_q$ for α_q.

It is easy to see that the extra work in the pivot step required by GDPO is generally small.

1. Ratio test: same work as with traditional dual.

2. The break points of the piecewise linear dual objective function have to be stored and sorted. This requires extra memory for the storage, and extra work to sort. However, the t_k values have to be sorted only up to the point where $f(t)$ reaches its maximum. Therefore, if an appropriate priority queue is set up for these values the extra work can be kept at minimum.

The dual phase-1 reduced costs can be calculated from (12) which means $Bv = \tilde{a}$ has to be solved for v, i.e., a further solution of a system of equations with B is required. This computational effort can be reduced in case of the traditional (‘first break point’) method if no more than one dual logical changes its feasibility status (several can change if the minimum ratio is not unique). In this case the corresponding variable can be chosen to enter the basis (thus its α_q becomes available) and q leaves either M or P. From (12) it follows that v can be updated in the former case as

$$\bar{v} = E\left(\sum_{j\in M\setminus\{q\}} \alpha_j - \sum_{j\in P} \alpha_j\right) = E(v - \alpha_q) \tag{15}$$

and in the latter case as

$$\bar{v} = E\left(\sum_{j\in M} \alpha_j - \sum_{j\in P\setminus\{q\}} \alpha_j\right) = E(v + \alpha_q). \tag{16}$$

If there is no change in sets M and P then $\bar{v} = Ev$.

Equations (15) and (16) suggest how v can be updated if GDPO is used. Namely, the change of the feasibility status of the dual logicals must be recorded (both sets can shrink or expand). The corresponding columns define composite vector $\tilde{a}'$ and

$$\alpha' = B^{-1}\tilde{a}' \tag{17}$$

in a similar way as in (12). Finally, the updated v is obtained as

$$\bar{v} = E(v - \alpha'). \tag{18}$$

If there are not too many changes in the feasibility status then $\tilde{a}'$ can be a rather sparse vector. Therefore, updating v using (17) and (18) can be computationally cheaper than recomputing v from its definition in (12).

3.2.3 Coping with degeneracy

In case of dual degeneracy, we have $d_j = 0$ for one or more nonbasic variables. If these positions take part in the ratio test they define 0 ratios. This leads to a (multiple) break point at $t = 0$. Choosing one of them results in a non-improving iteration. Assume the multiplicity of 0 is ℓ, i.e.,

$$0 = |t_1| = \cdots = |t_\ell| < |t_{\ell+1}| \le \cdots \le |t_Q|. \tag{19}$$

Denote the corresponding coefficients in the updated pivot row by $\alpha_{j_1}, \ldots, \alpha_{j_\ell}, \ldots, \alpha_{j_Q}$ (omitting the subscript p, for simplicity). The maximizing break point is defined by subscript k such that $s_k > 0$ and $s_{k+1} \le 0$, i.e.,

$$s_k = s_0 - \sum_{i=1}^{k} |\alpha_{j_i}| > 0 \quad \text{and} \quad s_{k+1} = s_0 - \sum_{i=1}^{k+1} |\alpha_{j_i}| \le 0.$$

If for this k relation $k > \ell$ holds then, by (19), we have $|t_k| > 0$. Hence, despite the presence of degeneracy, a positive step can be made towards dual feasibility. If $k \leq \ell$ then the step will be degenerate.

3.2.4 Implementation

For the efficient implementation of GDPO a sophisticated data structure (priority queue) is needed to store and (partially) sort the break points. Additionally, since row and column operations are performed on A, it is important to have a data structure that supports efficient row and columnwise access of A.

3.2.5 An example of the algorithmic step

We demonstrate the operation of GDPO on the following example. It is assumed that the pivot row p has been selected based on v_p and the updated pivot row has been determined. The types of the variables and the d_j values are given.

The problem has 8 nonbasic variables. Now $v_p = 10$ which is the $t \leq 0$ case. The sum of dual infeasibilities is $f = -35$. The solution is dual degenerate.

j	1	2	3	4	5	6	7	8	
type(x_j)	3	3	3	2	2	2	2	2	
Status									
α_{pj}	8	4	1	4	2	-1	-1	1	$v_p = 10$
d_j	-24	2	0	-8	-1	1	0	0	$f = -35$
Infeasibility	M	P		M	M				
Ratio	-3		0	-2	-0.5	-1	0		
2nd ratio	-3								

Altogether, 7 ratios have been defined, $Q = 7$. Note, for the first position there are two identical ratios. After sorting the absolute values of the ratios we obtain:

Index	1	2	3	4	5	6	7
j_k	7	3	5	6	4	1	1
$\lvert t_{j_k}\rvert$	0	0	0.5	1	2	3	3
α_{pj_k}	-1	1	2	1	4	8	8

Now, we can apply Step 7 of GDPO:

k	j_k	$\lvert t_k\rvert$	α_{pj_k}	$f_k = f_{k-1} + s_p^{k-1}(t_k - t_{k-1})$	$s_p^k = s_p^{k-1} - \lvert\alpha_{pj_k}\rvert$
0		0		-35	10
1	7	0	-1	-35	$10 - \lvert -1\rvert = 9$
2	3	0	1	-35	$9 - \lvert 1\rvert = 8$
3	5	0.5	2	$-35 + 8 \times (0.5 - 0) = -31$	$8 - \lvert 2\rvert = 6$
4	6	1	1	$-31 + 6 \times (1 - 0.5) = -28$	$6 - \lvert 1\rvert = 5$
5	4	2	4	$-28 + 5 \times (2 - 1) = -23$	$5 - \lvert 4\rvert = 1$
6	1	3	8	$-23 + 1 \times (3 - 2) = -22$	$1 - \lvert 8\rvert = -7$

We have used 6 of the 7 breakpoints. At termination of this step of GDPO $k = 6$, therefore, the entering variable is x_1 that has defined t_6. The dual steplength is -3 $(= t_6)$. Traditional methods would have stopped at the first breakpoint resulting in a degenerate iteration.

3.2.6 Key features of GDPO

The main computational advantage of the introduced GDPO algorithm is that it can make multiple steps with one updated pivot row. These steps correspond to several traditional iterations. A multiple step of GDPO requires very little extra work.

GDPO is a generalization of the traditional dual simplex algorithm in the sense that the latter stops at the smallest ratio (first break point) while GDPO can pass many break points making the maximum progress towards dual feasibility with the selected outgoing variable. Its efficiency of is not hampered by the presence of all types of variables.

GDPO possesses a potentially favorable anti-degeneracy property as shown in section 3.2.3.

The freedom of multiple choice created by the break points can be used to enhance the numerical stability of GDPO. Namely, if $|\alpha_{pj_q}|$ of the optimal breakpoint is too small and $q > 1$ we can take one of the previous breakpoints with sufficiently large pivot element.

GDPO works within the framework of the simplex method. It approaches a dual feasible solution (if one exists) by basis changes. It is a sort of a greedy algorithm. This is a locally best strategy. There is no guarantee that, ultimately, it will lead to a faster termination of phase-1 than other methods. As GDPO has some remarkable properties it is worth testing its actual performance on real world problems. In section 4 we report a limited computational experiment and comment the findings.

4 Computational experience

To get an idea how GDPO performs on real world problems we have performed a computational testing with an experimental implementation of the algorithm. As type-2 and type-3 variables are the 'difficult' ones we have chosen problems where the majority of the variables fall into these categories.

The first ten of the test problems were taken from the `netlib/lpdata` [5] set while the remaining three from other sources. The problem statistics are shown in Table 3.

Table 3: Statistics of problems included in computational testing of GDPO.

				Number of variables by type			
Problem	Rows	Columns	Nonzeros	Type-0	Type-1	Type-2	Type-3
25fv47	822	1571	11127	0	0	1571	0
bnl2	2325	3489	16124	0	0	3489	0
boeing1	351	384	3865	0	228	156	0
cycle	1903	2857	21322	0	77	2773	7
degen3	1504	1818	26230	0	0	1818	0
maros	847	1443	10006	35	0	1408	0
perold	626	1376	6026	64	266	958	88
pilot_we	723	2789	9218	78	294	2337	80
stair	357	467	3857	82	6	373	6
stocfor3	16676	15695	74004	0	0	15695	0
rentacar	6804	9557	42019	650	179	8728	0
scrs_3	16546	17420	71401	0	0	17420	0
unimin	5422	45569	168220	2	1449	44118	0

4.1 Test results

In the tests we first compared the performance of GDPO with the case when the first break point is taken. The number of iterations in dual phase-1 are shown in Table 4.

The tests were carried out with the following settings: no presolve, starting from the all logical basis and using the dual Dantzig pricing for determining the outgoing variable. The phase-1 reduced costs were recomputed in each iteration from the definition in (12).

Next, we investigated the behavior of GDPO with respect to the number of long steps in terms of the number of break points used per iteration. While the findings are rather interesting, it is not easy to demonstrate them in full detail. The issue of interest is the

Table 4: The number of dual phase-1 iterations until dual feasibility. Column 'Dual, $k = 1$' is for the 'first break point' method. Solution strategy: no presolve, all logical basis, Dantzig pricing.

	# of initial	# of Ph-1 itns with	
Problem	infeasibilities	GDPO	Dual, $k = 1$
25fv47	41	238	1033
bnl2	156	53	116
boeing1	164	9	102
cycle	498	128	653
degen3	1249	2096	4389
maros	162	666	799
perold	7	663	587
pilot_we	91	580	1065
stair	1	180	152
stocfor3	5077	10260	11848
rentacar	2	1778	1629
scrs_3	4355	12420	15556
unimin	43914	5265	50702

frequency of the number of break points used in the iterations. In Table 5 we present the length of steps in terms of the number of break points (q) used. The entries of the table show this frequency. For instance, 156 in the intersection of row of `degen3` and column headed by '4' says that there were 156 iterations when the 4th smallest ratio (4th break point) was used to determine the incoming variable. As there was a wide variation in the number of break points (header of the table), from 5 we indicate interval frequencies. While it makes the results displayable interesting details remain hidden. For instance, `boeing1` used 291 break points in one iteration which is counted in the '50+' column.

Table 5: Length of steps in terms of the number of break points. The entries give the number of iterations that used the number of break points shown in the header.

	Number of break points used								Iterations
Problem	1	2	3	4	5–10	11–20	21–50	50+	in phase-1
25fv47	80	76	38	16	24	4	–	–	238
bnl2	20	18	4	–	1	10	–	–	53
boeing1	1	–	–	–	5	2	–	1	9
cycle	49	24	11	11	14	8	11	–	128
degen3	15	83	212	156	1005	550	72	3	2096
maros	258	200	97	44	58	8	1	–	666
perold	352	138	60	22	60	18	11	2	663
pilot_we	308	105	44	26	79	50	12	6	630
stair	142	34	2	–	1	1	–	–	180
stocfor3	3730	3491	1604	761	989	44	1	–	10620
rentacar	1578	132	34	17	14	3	–	–	1778
scrs_3	8398	3913	92	17	–	–	–	–	12420
unimin	427	691	273	371	553	2736	214	–	5265

4.2 Discussion

First of all, it has to be noted that the dual may not be the ideal solution algorithm for the problems in the test. Also, the solution strategy used is very basic. However, these circumstances are unlikely to alter the tendencies. Therefore, the observed relative merits/drawbacks of GDPO may be more generally true. Obviously, we can expect a considerable reduction in the iteration counts if some more sophisticated procedures are used (presolve, advanced starting basis, steepest edge pricing, etc.).

The presented comparison was made between GDPO and the 'first breakpoint' strategy. As the dual reduced costs were recomputed and not updated the time per iteration was practically identical in the two algorithms. The variations were within the limit of 'white noise'. Apparently, the difference in the iteration speed due to some extra computations in the ratio test was not noticeable in the shadow of the organizational overhead of the iterations.

The results show a relatively wide spread. In three cases (`perold, stair, rentacar`) GDPO was slightly worse in terms of the number of iterations. In 6 cases (`bnl2, degen3, maros, pilot_we, stocfor3, scrs_3`) it was better by a considerable margin. In the remaining four other cases (`25fv47, boeing1, cycle, unimin`) GDPO performed significantly better. `boeing1` and `unimin` are particularly good examples where the potentials of GDPO materialized very spectacularly.

It is remarkable that GDPO really tends to use more breakpoints (see Table 5). The unseen advantage of it is a possibly more stable operation. During the test we noticed that some pivots were rejected by the algorithm and a better sized one was found that still contributed to the improvement of the phase-1 dual objective function.

5 Summary

We have presented a generalization of the dual phase-1 algorithms that can handle all types of variables efficiently. It is based on the piecewise linear nature of the defined dual phase-1 objective function. The main advantage is that a number of very cheap iterations can be made with one updated pivot row. As an additional benefit, GDPO possesses several favorable features making it a serious candidate for inclusion in optimization software.

We have shown GDPO can be implemented efficiently. Preliminary computational testing has given encouraging results. However, a more thorough computational study is required to better understand the benefits and possible weaknesses of GDPO.

As a last point, we indicate that for the selection of the pivot row any of the known methods can be used, including the Dantzig rule [2], dual Devex [7] or dual steepest edge [3].

6 Acknowledgements

The author wishes to express his gratitude to two anonymous referees. In particular, one of them pointed out how the original version of GDPO could be simplified. The current version of the paper greatly benefited from his/her helpful remarks.

References

[1] Chvátal, V., *Linear Programming*, Freeman and Co., 1983.

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

[3] Forrest, J.J., Goldfarb, D., "Steepest edge simplex algorithms for linear programming" *Mathematical Programming*, 57, 1992, No. 3., p. 341–374.

[4] Fourer, R., "Notes on the Dual Simplex Method", Unpublished, March, 1994.

[5] Gay, D.M., "Electronic mail distribution of linear programming test problems", *COAL Newsletter*, Mathematical Programming Society, 13, 1985, p. 10–12.

[6] Greenberg, H.J., "Pivot selection tactics", in Greenberg, H.J. (ed.), *Design and Implementation of Optimization Software*, Sijthoff and Nordhoff, 1978, p. 109–143.

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

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

[9] Maros, I., "A general Phase-I method in linear programming", *European Journal of Operational Research*, 23(1986), p. 64–77.

[10] Maros, I., "A Piecewise Linear Dual Procedure in Mixed Integer Programming", in F. Giannesi, R. Schaible, S. Komlosi (eds.), *New Trends in Mathematical Programming*, Kluwer Academic Publishers, 1998, pp. 159–170.

[11] Maros, I., Mitra, G., "Simplex Algorithms", Chapter 1 in Beasley J. (ed.) *Advances in Linear and Integer Programming*, Oxford University Press 1996, p. 1–46.

[12] Orchard-Hays, W., *Advanced Linear-Programming Computing Techniques*, McGraw-Hill, 1968.

[13] Padberg, M., *Linear Optimization and Extensions*, Springer, 1995.

[14] Wolfe, Ph., "A technique for resolving degeneracy in linear programming", *SIAM Journal of Applied Mathematics*, 11, 1963, p. 205–211.

[15] Wolfe, Ph., "The composite simplex algorithm", *SIAM Review*, 7 (1), 1965, p. 42–54.