Mathematical Programming 45 (1989) 437-474
North-Holland

A PRACTICAL ANTI-CYCLING PROCEDURE FOR LINEARLY CONSTRAINED OPTIMIZATION

Philip E. GILL
Department of Mathematics, University of California, San Diego, La Jolla, CA 92093, USA

Walter MURRAY and Michael A. SAUNDERS
Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA 94305, USA

Margaret H. WRIGHT
AT&T Bell Laboratories, Murray Hill, NJ 07974, USA

A procedure is described for preventing cycling in active-set methods for linearly constrained optimization, including the simplex method. The key ideas are a limited acceptance of infeasibilities in all variables, and maintenance of a "working" feasibility tolerance that increases over a long sequence of iterations. The additional work per iteration is nominal, and "stalling" cannot occur with exact arithmetic. The method appears to be reliable, based on computational results for the first 53 linear programming problems in the *Netlib* set.

Key words: Linear programming, simplex method, active-set methods, degeneracy, cycling.

1. Introduction

Degeneracy is often regarded as a discomforting but otherwise tolerable hindrance to the simplex method, and to other active-set algorithms for solving optimization problems involving linear constraints. Sequences of non-improving steps are known to occur (perhaps many times while solving a given problem), but such sequences are rarely observed to be infinite. The phenomenon of "stalling" is therefore recognized and accepted, but "cycling" is deemed very unlikely to occur.

In spite of such folklore, cycling remains a theoretical possibility, and a rigorous anti-cycling procedure can provide welcome peace of mind to users and implementors alike, particularly if the cost is small. Numerous authors have suggested anti-cycling techniques of differing flavors. A partial list includes Balinski and Gomory [1], Benichou et al. [3], Bland [4], Dantzig [6, 7], Dantzig et al. [8], Fletcher [12], Graves [22], Klotz [25], Rockafellar [35] and Wolfe [39]. The practical benefits

The material contained in this report is based upon research supported by the Air Force Office of Scientific Research Grant 87-01962; the U.S. Department of Energy Grant DE-FG03-87ER25030; National Science Foundation Grants CCR-8413211 and ECS-8715153; and the Office of Naval Research Contract N00014-87-K-0142.

of anti-cycling methods have been discussed recently by Ryan and Osborne [36] and Falkner [9].

An anti-cycling procedure will be described in this paper that involves little overhead and has proved to be effective in practice. Two features are crucial: controlled infeasibility of all variables (including nonbasics) and a "working" feasibility tolerance that increases slightly and consistently through an extended sequence of iterations. Sections 2 and 3 review background material about the simplex method and steplength selection. The new technique is introduced in Section 4, followed by presentation of a simplified version in Section 5. The relationship of the new approach to Wolfe's [39] method is considered in Section 6, and issues arising in Phase 1 are treated in Section 7. The use of the procedure in active-set methods for general linearly constrained problems is addressed in Section 8. Computational results are given in Section 9, and our conclusions are stated in Section 10.

2. Background

Most of our discussion will concern application of the *simplex method* [6] to the standard-form primal linear programming (LP) problem:

$$\begin{aligned} &\text{minimize} \quad c^T x \\ &\text{subject to} \quad Ax = b, \quad l \leq x \leq u, \end{aligned} \tag{2.1}$$

where $x \in \mathbb{R}^n$ and A is $m \times n$ ($m \leq n$). We first describe an "idealized" version of the simplex method (with exact arithmetic), and then consider issues that arise in implementation.

2.1. A typical iteration

In common with many optimization methods, the simplex method can usefully be interpreted as a sequence of *two-part* iterations. The current iterate x is assumed to be feasible, i.e., $Ax = b$ and $l \leq x \leq u$. If x is not optimal, a search direction $p \in \mathbb{R}^n$ is computed that reduces the objective function and remains locally feasible. A nonnegative scalar steplength α is chosen that specifies the distance to be moved along p, and the next iterate is defined as

$$x \leftarrow x + \alpha p. \tag{2.2}$$

At each iteration, the n variables x are explicitly divided into two disjoint sets: m *basic* variables x_B and $n - m$ *nonbasic* variables x_N. The m columns of A associated with x_B constitute B, a nonsingular $m \times m$ matrix called the *basis*, and the remaining $n - m$ columns are designated as N. The columns of B and N may occur anywhere in A and in any order, but the relation $Ax = Bx_B + Nx_N$ is valid at every iteration. The basis B is represented by certain matrix factors, which are updated at most

iterations. These factors are computed from scratch at the first iteration and periodically thereafter.

The *pricing* or *column-selection* strategy chooses a single nonbasic variable (the *pivot column*) that can be moved in a feasible direction while reducing the objective function. This decision is based on reduced costs (Lagrange multipliers) for the nonbasic variables, which indicate whether a move away from the current value will reduce the objective function. (The chosen nonbasic variable usually enters the basis.) All nonbasic variables remain unchanged except the chosen one. To ensure that the new iterate satisfies $Ax = Bx_B + Nx_N = b$, the components of p corresponding to basic variables are assigned so that $Ap = 0$.

The steplength α is chosen to retain feasibility of the next iterate. This part of a simplex iteration (selecting the *pivot row*) is central to the new anti-cycling technique, and will be discussed in detail in Section 3.

The simplex method can also be described in the more general language of *active-set methods* for general linearly constrained optimization (see, e.g., [11, 21]), which follow the generic iteration model (2.2). In an active-set method, a *working set* of constraints (usually including constraints that are currently active) is defined at each iteration, and constraints are deleted from and added to this set as the iterations proceed. Constraint deletion is based on signs of Lagrange multiplier estimates for constraints in the working set, and affects the definition of p; constraint addition is guided by the need to remain feasible, and specifies α. For the special case of the simplex method, constraint deletion and addition are equivalent to selection of the pivot column and row. Since the discussion of this paper concentrates primarily on the simplex method but applies to general active-set methods as well, we shall occasionally switch terminology, or describe the same process from these two viewpoints.

2.2. Representation of nonbasic variables

For many years, implementations of the simplex method tended to treat the *numerical* values of the basic and nonbasic variables in different ways. The basic variables satisfy the linear system

$$Bx_B = b - Nx_N, \tag{2.3}$$

and hence their values can be computed from x_N and a factorization of B. In contrast, nonbasic variables are usually assumed to be at one of their bounds. This property can be achieved by *implicitly* assigning the exact value of the appropriate bound to each nonbasic variable (using a status indicator).

Elementary presentations of the simplex method typically include only nonnegativity restrictions, in which case the bounds are zero and infinity ($0 \leq x_j \leq \infty$) and nonbasic variables are implicitly zero. When general bounds l_j and u_j are allowed, the value of a nonbasic variable is usually defined to be one of the bounds; thus, x_j is either l_j or u_j, depending on which bound is marked as "active". A complication arises with a "free variable" (corresponding to $l_j = -\infty$, $u_j = +\infty$), which may need

to be treated as nonbasic at an intermediate iteration even though it is likely to be basic at a solution. A nonbasic free variable is usually defined to be *zero*, thereby avoiding the need to store any other value. (This approach was used in various versions of MINOS up to and including MINOS 5.0 [27].)

History aside, several practical benefits arise if nonbasic variables are not required to equal one of their bounds. For example, nonbasic variables can be initialized at a "safe" value (say, zero) if both specified bounds are inordinately large (e.g., $l_j = -10^8$, $u_j = 10^8$). Similar advantages arise when restarting modified problems and recovering from singular bases. In MINOS 5.1 [28], explicit values are stored for all variables, and a nonbasic variable is allowed to take any value within its bounds.

Even when freedom is granted to assume values other than their original bounds, the fundamental algorithmic role of nonbasic variables is unaltered: they *remain fixed* at a particular iteration, i.e., the *change* in nonbasic variables (except one) is constrained to be exactly zero.

For active-set methods, the analogous concept is to include constraints in the working set that are not necessarily active at one of their original bounds. This idea is expressed in the "pseudo-constraints" of Fletcher and Jackson [14], the "artificial constraints" of Gill and Murray [19], and the "pegged variables" of Nazareth [29, 30].

2.3. *Traditional treatment of feasibility*

The simplex method terminates "normally" in three situations: a bounded optimal solution has been found; no feasible point exists; or the objective function is unbounded below in the feasible region. A bounded LP solution must satisfy two criteria: *optimality* and *feasibility*. With exact arithmetic, only the optimality test needs to be applied after an initial feasible point is found, since simplex iterates are constructed to remain feasible. In practice, however, both criteria must be interpreted numerically, based on optimality and feasibility *tolerances*, which are normally much larger than machine precision. For example, if machine precision (denoted throughout by ε) is 10^{-16}, a typical feasibility tolerance δ_f is 10^{-6}.

The optimality tolerance is used in a straightforward fashion to judge whether the reduced costs for the current set of nonbasic variables are sufficiently positive or negative. Testing for feasibility is more complicated. When nonbasic variables implicitly satisfy their bounds as described above, their values need not be checked. Basic variables, however, are *computed* using (unavoidably) inexact arithmetic, and hence may violate their bounds. When B is refactorized in a numerically stable fashion, standard error analysis implies that $Ax = b$ will be satisfied within a tolerance that involves a multiple of machine precision [37]. After refactorization, the following feasibility test for the basic variables is applied:

$$l_B - \delta_f e \leq x_B \leq u_B + \delta_f e, \tag{2.4}$$

where $\delta_f > 0$ is the feasibility tolerance and e is a vector of ones. If this test is not satisfied, Phase 1 of the simplex method is invoked to move any infeasible variables

toward their violated bounds; otherwise, Phase 2 starts (or resumes). A traditional computed "optimal" solution therefore satisfies the *original* bounds $l_N \leq x_N \leq u_N$ for the nonbasic variables, and the relaxed bounds (2.4) for the basic variables.

A key feature of the new anti-cycling procedure is that *nonbasic* variables are allowed to violate their bounds (see Section 4). In practice, few (if any) nonbasic variables will be infeasible at the final solution.

3. Selection of the steplength

In the simplex method, the value of the steplength α (see (2.2)) is chosen by the *row-selection* procedure, which also delivers the index r of a *blocking variable* that becomes nonbasic. (We sometimes refer to the index r itself as the blocking variable.) The usual relationship between α and r is that variable r exactly reaches one of its bounds at $x + \alpha p$. (If α is infinite, the objective function is unbounded below in the feasible region.) The *pivot element* in the simplex method is p_r, the component of p corresponding to the blocking variable. A rule of thumb is that "small" pivots lead to ill-conditioned basis matrices.

From an active-set viewpoint, a *blocking constraint* is chosen to be added to the working set, and α is usually the step at which the blocking constraint becomes active. The condition of the working set deteriorates if the blocking constraint is "almost" linearly dependent on constraints already in the working set.

In this section, we discuss various strategies for choosing α and r.

3.1. Computational procedures

Determination of the steplength α involves two distinct computational procedures, which we now state in algorithmic "pseudo-code", using simplex terminology. These procedures will be invoked with different arguments throughout the remainder of the paper.

For each variable that may encounter a bound, the step along p to that bound must be calculated. This computation is summarized in the function step shown below. The formal parameters are χ (the "current point"); ρ (the "search direction"); l (a lower bound, which may be $-\infty$); ν (an upper bound, which may be $+\infty$); and t, a nonnegative tolerance that defines a "negligible" value of ρ. The value of step is defined as follows. If ρ is negative and non-negligible, step gives the multiple of ρ that "reaches" l when added to χ, i.e., $\chi + \text{step} \times \rho = l$. If ρ is positive and non-negligible, step gives the analogous multiple that reaches ν. If ρ is negligible, or if the relevant bound is infinite in magnitude, step is $+\infty$.

```
function step(χ, ρ, l, ν, t);
if ρ < −t and l > −∞ then step ← (l − χ)/ρ
  else if ρ > t and ν < +∞ then step ← (ν − χ)/ρ
  else step ← +∞
end if
```

The procedure ratio_test given below performs the "minimum ratio test". (In active-set terms, ratio_test calculates the "step to the nearest constraint".) Given an n-vector x, a search direction p, lower and upper bound arrays low and up, and a tolerance t, ratio_test returns a steplength α_{rat} and the index j_{rat} of a blocking variable.

The value of step is calculated for each variable as specified above. The *smallest* value of step is designated as α_{rat}, and j_{rat} is the index of a corresponding variable. If $j_{\text{rat}} = 0$, there is no blocking variable (i.e., no variable reaches a bound for any finite positive step along p.) We shall abuse standard programming convention slightly by writing

$$(\alpha_{\text{rat}}, j_{\text{rat}}) = \text{ratio\_test}(x, p, \text{low}, \text{up}, t)$$

to mean that α_{rat} and j_{rat} are assigned the values calculated in the procedure.

procedure ratio_test(x, p, low, up, t);
$j_{\text{rat}} \leftarrow 0$; $\alpha_{\text{rat}} \leftarrow +\infty$;
for $j = 1$ **until** n **do**
  $\alpha_j \leftarrow \text{step}(x_j, p_j, \text{low}_j, \text{up}_j, t)$;
  if $\alpha_j < \alpha_{\text{rat}}$ **then**
    $\alpha_{\text{rat}} \leftarrow \alpha_j$; $j_{\text{rat}} \leftarrow j$
  end if
end for

When the tolerance t is omitted from the parameters of step or ratio_test, it should be taken as *zero*.

3.2. The textbook and Harris ratio tests

If x satisfies $l \leq x \leq u$, the *textbook* ratio test defines α and r according to

$$(\alpha, r) = \text{ratio\_test}(x, p, l, u). \tag{3.1}$$

With this choice, α is the largest step that keeps $x + \alpha p$ feasible, and r is the index of the variable that reaches its bound at α. If α is finite, the value of x_r at the end of the iteration is thus l_r or u_r (depending on the sign of p_r), and all other variables continue to satisfy their bounds.

The textbook ratio is "ideal" in the sense that it guarantees the maximum reduction in the objective function while retaining feasibility at the next iterate. In practice, however, numerical difficulties (typically, an ill-conditioned basis) result when the pivot element p_r is "too small". Although one might hope that step(x_j, p_j, l_j, u_j) will be large when $|p_j|$ is small (so that j will not be chosen as the blocking variable), it can be small if x_j happens to be very close to the relevant bound. Unless there is a tie for the blocking variable, the textbook ratio test offers no mechanism for avoiding small pivots.

In [24], Harris suggested a technique intended to encourage the selection of blocking variables with larger pivots, at the price of allowing infeasibilities in the basic variables. The Harris strategy is a *two-pass* procedure. The first pass determines a "relaxed" steplength $\alpha 1$ by *enlarging* the original bounds:

$$(\alpha 1, r1) = \text{ratio\_test}(x, p, l - \delta e, u + \delta e), \tag{3.2}$$

where δ is a feasibility tolerance. To ensure that $\alpha 1 \geq 0$, x must satisfy the perturbed bounds $l - \delta e \leq x \leq u + \delta e$. If $l \leq x \leq u$, a step of $\alpha 1$ along p cannot violate any individual bound by more than δ. Hence, $\alpha 1$ is taken as an *upper bound* on the allowable step. The second pass chooses the blocking variable from among all variables for which the "textbook" step (to the *exact* bound) does not exceed $\alpha 1$, giving a second step $\alpha 2$:

$$r = \arg\max_j \{|p_j|\} \quad \text{for } j \text{ such that } \text{step}(x_j, p_j, l_j, u_j) \leq \alpha 1, \tag{3.3a}$$

$$\alpha 2 = \text{step}(x_r, p_r, l_r, u_r). \tag{3.3b}$$

The value of α is then $\max\{\alpha 2, 0\}$, i.e., α is taken as $\alpha 2$ if $\alpha 2$ is nonnegative, and as zero otherwise. The blocking variable is r in either case.

To see how these tests differ, consider two basic variables $x = (0.0009, 1)^{\mathrm{T}}$ subject to nonnegativity bounds with $p = (-0.1, -100)^{\mathrm{T}}$ and feasibility tolerance $\delta = 10^{-3}$. Using the textbook procedure, we have

$$\text{step}(x_1, p_1, l_1, u_1) = 0.009 \quad \text{and} \quad \text{step}(x_2, p_2, l_2, u_2) = 0.01. \tag{3.4}$$

The smaller step corresponds to the first variable, which would be chosen as the blocking variable.

In contrast, execution of the first pass (3.2) of the Harris procedure gives

$$\text{step}(x_1, p_1, l_1 - \delta, u_1 + \delta) = 0.019$$

and

$$\text{step}(x_2, p_2, l_2 - \delta, u_2 + \delta) = 0.01001,$$

so that $\alpha 1 = 0.01001$. In the second pass, both values of step calculated in (3.3a) are less than $\alpha 1$ (see (3.4)). Since $|p_2|$ is larger than $|p_1|$, $r = 2$ and $\alpha 2 = 0.01$. The result is that $\alpha = \alpha 2 = 0.01$, with variable two becoming nonbasic and

$$x + \alpha p = \begin{pmatrix} 0.0009 \\ 1 \end{pmatrix} + 0.01 \begin{pmatrix} -0.1 \\ -100 \end{pmatrix} = \begin{pmatrix} -0.0001 \\ 0 \end{pmatrix}.$$

Notice that a larger pivot has been chosen by the Harris procedure, but that the first variable (which remains basic) now violates its lower bound by 0.0001—an amount no greater than δ.

Figure 1 illustrates the contrast between the textbook and Harris procedures in an active-set context. As we move from the point labeled "x" along the horizontal constraint as indicated by the arrow, four constraints (marked 1-4) intersect the path; each constraint is shaded on the *infeasible* side. The path generated by the textbook ratio test is x-a-b-c-d, and all iterates remain feasible with respect to the

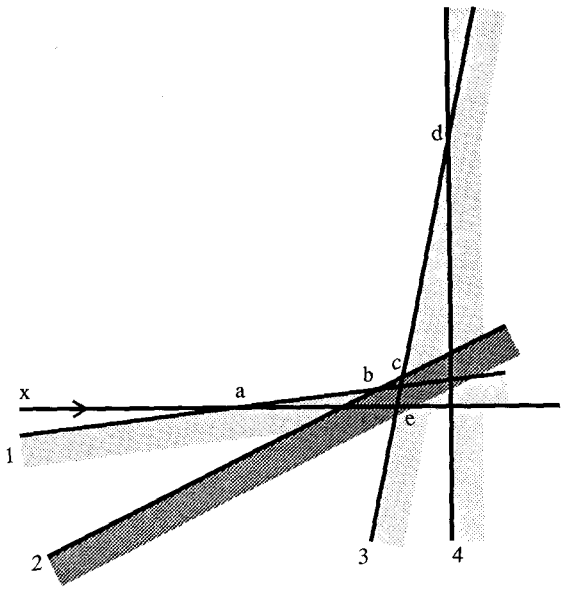

Fig. 1. The textbook and Harris procedures.

original constraints. In contrast, the Harris procedure first finds the closest intersection of a *perturbed* constraint (see (3.2)). In the figure, the width of shading indicates the feasibility tolerance. The darker shading for constraint 2 indicates that its perturbed intersection with the horizontal line is nearest to x, and $\alpha 1$ is the step from x to the intersection of the outer edge of the darker shading and the horizontal line. The second pass (3.3) determines that the *exact* versions of constraints 1, 2 and 3 intersect the horizontal to the left of $\alpha 1$. The "least oblique" among these is constraint 3, which means that $r2 = 3$ and $\alpha 2$ is the step to the vertex labeled "e". The path taken by the Harris procedure is thus x-e-d, where constraints 1 and 2 are slightly violated at the second iterate.

3.3. *A consequence of infeasible basic variables*

The Harris test necessarily allows infeasibilities in each basic variable. A less obvious consequence is that the iterates may fail to satisfy $Ax = b$ *if nonbasic variables are treated in the traditional way.* To see why, consider the case when $x_1 = -0.001$, $p_1 = -1$ and $l_1 = 0$. Since $\text{step}(x_1, p_1, l_1, u_1)$ is *negative*, the value of $\alpha 2$ defined by (3.3b) could also be negative. If $\alpha 2 < 0$, the Harris procedure sets $\alpha = 0$ but retains the same blocking variable x_r, which then *becomes nonbasic.* With a traditional implementation of the simplex method, the value of the newly nonbasic x_r would actually be *changed* (perhaps implicitly) to move it exactly onto its bound (see Section 2.2). Assuming that δ is the current feasibility tolerance, moving x_r onto its bound is equivalent to performing an *extra* step in which the current iterate x is replaced by $x + \mu e_r$, where e_r is the rth coordinate vector and $|\mu| \leq \delta$. Such a change

can produce an error of order δ, i.e., much larger than machine precision, in satisfying $Ax = b$.

In practice, errors of this kind tend to be eliminated each time the basis is refactorized, since the basic variables are typically recomputed using (2.3) in order to satisfy $Ax = b$ accurately. Provision is made to return to Phase 1 if the recomputed variables lie outside their bounds by more than δ. On well-behaved problems, few iterations (if any) are required to regain feasibility, but in runs lasting thousands of iterations, the risk of a few extra iterations between refactorizations (typically, every 50 iterations) amounts to a nontrivial overhead. In the worst case, "few" can be more than the refactorization frequency and an optimum may not be achieved. When solving problems with a nonlinear objective function, a perturbation of order δ in what should theoretically remain the same point may cause failure in the linesearch because of the resulting discontinuity.

A simple way to avoid this difficulty is to implement a "zero" step *literally*. With such an approach, a slightly infeasible blocking variable becomes nonbasic, but *its infeasible value is retained* rather than moving onto its bound. The variable is temporarily frozen at that value (across basis factorizations if necessary) until the normal pricing strategy allows it to move. Provision can still be made to revert to Phase 1 after refactorization; given a stable basis-handling package, however, the likelihood of losing feasibility is greatly reduced.

An alternative is to allow a *negative step* whenever $\alpha 2$ of (3.3b) is negative, giving the blocking variable a chance to move exactly onto its bound. This approach has been used in the quadratic programming and linear least-squares codes QPSOL 3.2 and LSSOL 1.0 [20, 18]. However, it is then necessary to perform a ratio test on the reverse search direction $-p$, obtaining a possibly *different* blocking variable that again may be unable to reach its bound exactly. Since the objective value will move slightly in the wrong direction, care must be taken to avoid entering an infinite loop.

A further alternative is to include the new anti-cyling technique, which is described next.

4. The EXPAND procedure

Our anti-cycling strategy is called the EXPAND procedure (EXPanding-tolerance ANti-Degeneracy procedure). The descriptor "expanding" is used because a working feasibility tolerance is maintained that increases slightly at the start of every iteration.

4.1. Motivation and definition

In conventional LP terminology, the EXPAND procedure is a *row-selection* method that specifies the choice of pivot row in the simplex method. The "maximum pivot" property of Harris's row-selection method [24] is retained, and permitting infeasibility in nonbasic variables removes the difficulty described in Section 3.3 with traditional implementations of the Harris procedure.

It should be emphasized that infeasibilities in the nonbasic variables are allowed only to improve numerical stability and reliability, not to achieve a greater reduction in c^Tx in an enlarged feasible region. Although the constraints

$$Ax = b \quad \text{and} \quad l - \delta_f e \leq x \leq u + \delta_f e \tag{4.1}$$

can always be satisfied when $Ax = b$ is compatible and δ_f is sufficiently large, it is desirable to terminate with a solution that is as close as possible to feasibility for the *unperturbed* problem (2.1). Since the final B-N partition is unpredictable, we anticipate that practitioners accustomed to (2.4) will find (4.1) essentially equivalent.

With exact arithmetic, classical cycling cannot occur in an algorithm of the form (2.2) if $c^Tp < 0$ and $\alpha > 0$ at each iteration, since the objective function strictly decreases. In our procedure, a positive value of α is ensured by enlarging the bounds on all variables slightly at *every* iteration. Let $\tilde{\delta}$ denote the "old" tolerance from the previous iteration. The "current" tolerance δ is defined as

$$\delta = \tilde{\delta} + \tau, \quad \text{where } 0 < \tau \ll \tilde{\delta}, \tag{4.2}$$

and hence is strictly larger than $\tilde{\delta}$. The result is that any point x satisfying $l - \tilde{\delta}e \leq x \leq u + \tilde{\delta}e$ must lie *strictly inside* the expanded bounds $(l - \delta e, u + \delta e)$, and a positive step may be taken in *any direction* before encountering a bound.

The EXPAND procedure is summarized by the following pseudo-code:

procedure EXPAND($x, p, l, u, t, \delta, \tau$);
$(\alpha 1, r1) \leftarrow$ ratio_test($x, p, l - \delta e, u + \delta e, t$); (first pass)
$r \leftarrow 0$; $p_{\max} \leftarrow 0$;
for $j = 1$ **until** n **do** (second pass)
  $\alpha_j \leftarrow \text{step}(x_j, p_j, l_j, u_j, t)$;
  if $\alpha_j \leq \alpha 1$ **and** $|p_j| > p_{\max}$ **then**
    $r \leftarrow j$; $\alpha 2 \leftarrow a_j$; $p_{\max} \leftarrow |p_j|$
  end if
end for
$\alpha_{\min} \leftarrow \tau / |p_r|$; (minimum acceptable step)
$\alpha \leftarrow \max\{\alpha 2, \alpha_{\min}\}$.

As with ratio_test, we write

$$(\alpha, r) = \text{EXPAND}(x, p, l, u, t, \delta, \tau)$$

to mean that α and r are assigned the values computed during execution of the procedure.

The EXPAND procedure contains two passes that define $\alpha 1$, r and $\alpha 2$ exactly as in (3.2) and (3.3) of the Harris procedure. Because $\delta - \tilde{\delta} = \tau > 0$ and x satisfies $l - \tilde{\delta}e \leq x \leq u + \tilde{\delta}e$, $\alpha 1$ must satisfy

$$\alpha 1 \geq \tau / |p_{r1}|. \tag{4.3}$$

Since r corresponds to the largest pivot element among all variables for which the step to the exact bound does not exceed $\alpha 1$, $\alpha_{\min}$ must be positive and cannot exceed $\alpha 1$.

The crucial difference from the Harris procedure arises from imposition of the positive lower bound $\alpha_{\min}$ on the steplength. Assume that there is only one possible blocking variable x_r, and let Δ be the step from x_r to the corresponding bound: $\Delta = x_r - l_r$ or $u_r - x_r$. There are three cases.

Case 1. If $\Delta \geq \tau$, $\alpha 2$ is positive and exceeds $\alpha_{\min}$. With both the Harris and EXPAND procedures, α is taken as $\alpha 2$ (a "nondegenerate" step) and the blocking variable reaches its bound exactly.

Case 2. If $0 \leq \Delta < \tau$, x_r is feasible, but is closer than τ to its bound. The value of $\alpha 2$ is nonnegative, and the Harris procedure would move x_r onto its bound. With the EXPAND procedure, however, x_r is moved a step of $\alpha_{\min}$, and its bound becomes *violated* (by at most τ).

Case 3. If $\Delta < 0$, x_r is infeasible at the beginning of the iteration. The value of $\alpha 2$ is *negative* in this case, and the Harris procedure would take a step of *zero.* Depending on the treatment of nonbasic variables, x_r would either be moved onto its bound (see Section 2.2) or left unchanged at its present value. In contrast, the EXPAND procedure again takes a step of $\alpha_{\min}$, so that x_r becomes *more* infeasible. Even so, the new value of x_r cannot violate its bound by more than δ, i.e., an increase of τ from the maximum possible infeasibility at the beginning of the iteration.

A step of $\alpha_{\min}$ (Cases 2 and 3) corresponds to a "degenerate" step in which the blocking variable moves a total distance of τ and violates its bound at the end of the iteration. Since it is common for blocking variables to become basic at a later iteration, the total number of nonbasic infeasibilities at any stage is generally less than the number of degenerate steps so far.

Figure 2 illustrates Cases 1-3 (a normal step and two degenerate steps) for the EXPAND procedure. We assume that $p_r < 0$, so that x_r is constrained by its lower bound l_r, which is shown as the horizontal axis. The sloping arrows plot the value of $x_r + \alpha p_r$ against α, with three possible starting values for x_r. The shaded horizontal distance is $\alpha_{\min}$ in all cases, and the intersection point of the sloping arrow with the horizontal axis is $\alpha 2$. In the first case, x_r is not "close to" l_r, $\alpha 2$ exceeds $\alpha_{\min}$,

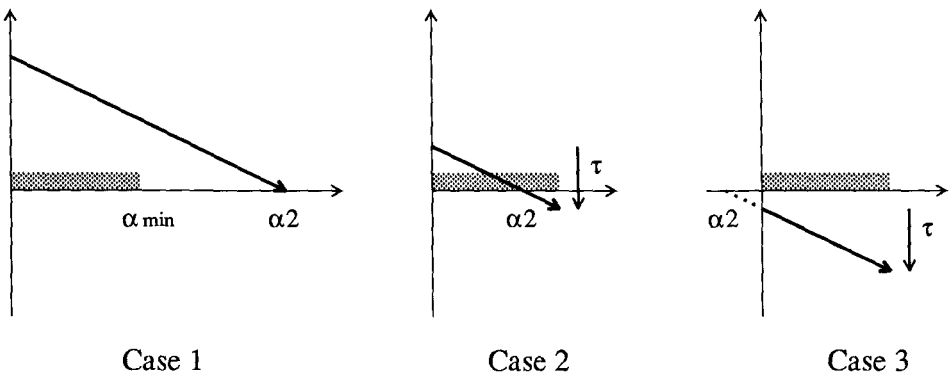

Fig. 2. Three configurations of the blocking variable.

the step of $\alpha 2$ is taken, and x_r moves exactly to its bound. In the middle case, x_r is feasible, but $\alpha 2$ is less than $\alpha_{\min}$. A step of $\alpha_{\min}$ moves x_r to an infeasible value. In the third case, x_r already violates its lower bound, $\alpha 2$ is *negative*, but again a step of $\alpha_{\min}$ is taken. Although x_r becomes more infeasible, the additional violation is limited by τ. In the second and third cases, the vertical arrow labeled "τ" shows the minimum required change in x_r.

4.2. *Increasing the feasibility tolerance*

We have just seen that the lower bound $\alpha_{\min}$ plays a special role in the EXPAND procedure in the "inner" context of a single simplex iteration. The equally important concept of *expansion* (increasing the feasibility tolerance) can be understood only when the EXPAND procedure is viewed as part of a *sequence* of simplex iterations.

Within this "outer" context, the procedure is invoked at iteration k to choose the steplength α_k and blocking variable r_k. The expansion of the feasible region is controlled by a "current" or "working" feasibility tolerance δ_k that monotonically increases at each iteration. The value of δ_k is defined in terms of the preceding tolerance δ_{k-1} using (4.2), and serves as the parameter δ of EXPAND. The following pseudo-code illustrates how the kth simplex iteration increases δ_k and calls EXPAND.

Compute the search direction p_k;
$\delta_k \leftarrow \delta_{k-1} + \tau$;
$(\alpha_k, r_k) = \text{EXPAND}(x_k, p_k, l, u, t, \delta_k, \tau)$;
$x_{k+1} \leftarrow x_k + \alpha_k p_k$;
Modify the sets of basic and nonbasic variables;
$k \leftarrow k+1$.

Since the iterates could in the worst case become more infeasible at every iteration, divergence is prevented by placing an upper bound K on the number of simplex iterations allowed before invoking a *resetting* procedure, to be described in the next section. A "master" feasibility tolerance δ_f is defined, with the essential property that $\delta_k < \delta_f$ for $k \leq K$. A sequence of at most K consecutive simplex iterations in which the value of δ_k successively increases will be called an *expanding sequence.*

The "best" choices for δ_f, δ_0 and K depend on the nature of the problem and on the machine precision. Given a "reasonable" precision ε, the following values are recommended:

$\delta_f = \varepsilon^{3/8}$ is the "master" feasibility tolerance;
$K = \varepsilon^{-1/4}$ is the maximum number of simplex iterations allowed before resetting;
$\delta_0 = 0.5\delta_f$ is the feasibility tolerance used to initiate an expanding sequence;
$\delta_K = 0.99\delta_f$ is the maximum feasibility tolerance during an expanding sequence;
$\tau = (\delta_K - \delta_0)/K$ is the amount by which δ_k increases at every iteration.

The philosophy reflected by these choices is to begin each expanding sequence with a value of δ_0 "similar" to δ_f, and to increase δ_k *slowly* through a *long* sequence of iterations.

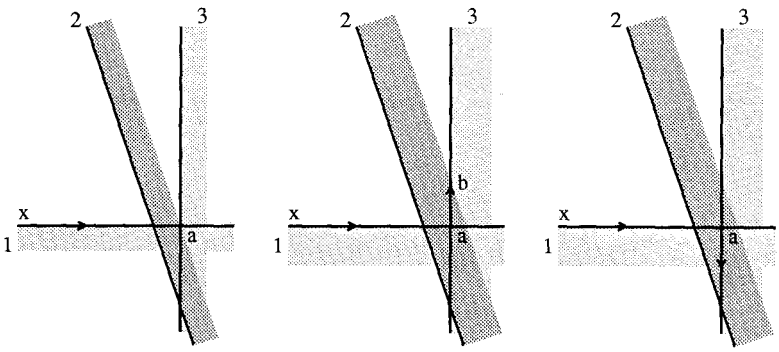

Fig. 3. An effect of increasing the feasibility tolerance.

For a machine with 16 decimal digits of precision ($\varepsilon = 10^{-16}$), the recommended values are $\delta_f = 10^{-6}$, $K = 10000$ and $\tau = 0.49 \times 10^{-10}$.

A crucial effect of increasing the feasibility tolerance is illustrated in Figure 3. In the leftmost figure, the current search direction moves to the right from the point labeled "x" along the horizontal constraint 1. The width of the shading is the current feasibility tolerance δ. The value α_1 from the first pass of EXPAND is the step to the point marked "a", which is (in this special case) both the intersection of constraints 1 and 3, *and* the intersection of constraint 1 with the perturbed version of constraint 2. Because constraint 3 is "less oblique", it is chosen as the blocking constraint, and the point a is the new iterate. Now consider the next iteration. If the search direction were "up" along constraint 3, as shown in the middle figure, no further movement would be possible *if the feasibility tolerance remained unaltered*, since constraint 2 is already violated by δ at a. However, because the feasibility tolerance is increased as shown by the wider shading, a *positive* step can be taken to the point labeled "b", where constraint 2 is violated by $\delta + \tau$ and becomes the blocking constraint. (In contrast, if the search direction were "down" as shown in the rightmost figure, tolerance of increased infeasibility with respect to constraint 2 would not affect the step.)

4.3. Resetting

After tolerating potentially increasing infeasibility over a long sequence of iterations, we require a "resetting" procedure to restore nonbasic variables to their bounds. The main steps are as follows.

(1) Every nonbasic variable that lies within δ_f of a bound is moved exactly onto the bound. (This set will include some of the variables that were slightly infeasible when they last became nonbasic.) A count is kept of the number of "nontrivial" adjustments, where "nontrivial" means (say) "greater than $\varepsilon^{2/3}$".

(2) If the count is positive, the basic variables are recomputed in terms of the newly adjusted nonbasic variables, so that $Ax = b$ will be satisfied to (essentially) machine precision.

(3) The feasibility tolerance is reinitialized to δ_0, and a new expanding sequence begins.

A reset occurs at the end of an expanding sequence, i.e., after K iterations. Following the reset, the optimizer resumes in Phase 1 or Phase 2 depending on the value of the feasibility tolerance at the *first* iteration, namely $\delta_1 = \delta_0 + \tau$.

Resetting is also associated with a tentative decision to *terminate* (i.e., the current point is apparently optimal, no feasible point can be found, or the solution appears to be unbounded). In our implementation, the reset procedure may be executed at most R times in this situation, where the choice of R depends on several factors. For well-conditioned problems, termination tests are typically satisfied (again) immediately after a reset, so that a value of $R = 1$ leads to little additional work and ensures "conventional" feasibility in the sense that no nonbasic variables lie outside their bounds.

In badly conditioned cases, on the other hand, a very large number of iterations may be needed to regain feasibility and optimality following a reset. To improve the probability of terminating at a "conventional" solution, it may be advisable to let $R = 2$, since a second reset normally adjusts fewer nonbasics than the first.

Finally, it may be appropriate to skip resetting entirely, i.e., to let $R = 0$. After k iterations of an expanding sequence, the current iterate is guaranteed to be feasible to within δ_k (assuming that Phase 1 has terminated). Since $\delta_k < \delta_f$, the current iterate satisfies (4.1) and may be considered acceptable as it stands.

This justification for choosing $R = 0$ might also apply in the degeneracy-resolving procedure of Benichou et al. [3, pp. 292-294], in which a perturbation is added to the right-hand side vector b. Once the perturbed problem has been solved, the perturbation is removed and the dual simplex algorithm is applied (often requiring no further iterations). If this approach were implemented with a perturbation of order δ_f (rather than the much larger perturbation suggested in [3]), the solution to the *perturbed* problem could be accepted for all practical purposes.

4.4. Convergence

The EXPAND procedure ensures a strict decrease in the objective function at each iteration within an expanding sequence. Because of resetting after K iterations, however, the possible increase in the objective after restoration of feasibility could theoretically lead to a classical cycle of period K. Our recommendation that K be very large is intended to make the probability of such a cycle essentially "negligible".

To emphasize the point, we note that previous implementations of the simplex method have been operating (in effect) with K set to the basis refactorization frequency—typically less than 100. Failures attributed to cycling have been rare (though not completely absent; for example, see [3, pp. 292-294]), and various other implementation details were probably contributing factors.

To a large extent, the chance of failure to converge due to resetting depends on $\text{cond}(B)$, the condition number of a typical basis matrix. If $\text{cond}(B)$ approaches $1/\varepsilon$, then errors in the computed solution can (in the worst case) be so large that

any algorithm may be unable to produce a solution that satisfies both feasibility and optimality requirements. However, there should be essentially no risk of failure with resetting if cond(B) approaches $1/\delta_f$, assuming that δ_f and K have "sensible" values.

Of course, it is impossible to choose a value of K that guarantees convergence in all cases. At the time of writing, some large examples of a particular class of (randomly generated) minimax problems have been found to enter a cycle of length 10 000 [16]. In one case, increasing K to 1 000 000 allowed an optimum to be reached after 15 000 iterations. Ill-conditioning was evident, but this serves to show the existence of pathological but solvable examples.

4.5. *The effect of ignoring small elements of p*

The tolerance t in the parameters of step is intended to provide a simple numerical safeguard against pivots that are "too small", by defining step as infinite when a component of p is "negligible" (see Section 3.1). (In MINOS, t is defined as $\varepsilon^{2/3}$ for linear programs and $\varepsilon^{2/3}\|p\|$ for nonlinear programs.) If t is positive, a variable x_j for which $|p_j|$ is "too small" *cannot be a blocking variable.* Unfortunately, this property may conflict with the assumptions that underlie the EXPAND procedure.

To ensure that the new iterate $x+\alpha p$ does not violate any bound by more than δ when $\alpha = \alpha_{\min}$, it is essential for x to lie *at least* τ away from its bounds $(l-\delta e, u+\delta e)$. If $\alpha|p_j| > \tau$ for any ignored element p_j, $x_j+\alpha_j p_j$ may violate its bound by as much as $\alpha|p_j|-\tau$.

In such cases, a simple precaution would be to test whether any components of $x+\alpha p$ violate the bounds $(l-\delta e, u+\delta e)$. Any that do could be moved onto those bounds. Alternatively, a smaller value of t could be chosen, which means that fewer elements of p are ignored. However, it is common for many elements of p to be very small, and excluding such elements from the ratio test can give significant computational savings on large problems.

Neither form of precaution has been included to date in our implementation. Such safeguards could be important if δ_f and τ were substantially different from the recommended values.

5. A simplified procedure

A preliminary version of the EXPAND procedure was used in the experiments conducted by Lustig [26]. This version was simpler and potentially more efficient on nondegenerate problems; we therefore summarize it in the pseudo-code below. As before, we assume that the feasibility tolerance has just been increased to $\delta = \tilde{\delta} + \tau$.

procedure SIMEXPAND($x, p, l, u, t, \delta, \tau$);
($\alpha 1, r1$) ← ratio_test(x, p, l, u, t); (first pass)
$\alpha_{\min} \leftarrow \tau/|p_{r1}|$; (minimum acceptable step)

if $\alpha 1 \geq \alpha_{\min}$ **then** $\alpha \leftarrow \alpha 1$; $r \leftarrow r1$;
  else $(\alpha, r) \leftarrow$ ratio_test$(x, p, l-\delta e, u+\delta e, t)$ (second pass)
end if

In contrast to EXPAND, this approach *reverses* the two passes in the Harris procedure, adding perturbations in the second pass. It has the advantage of terminating frequently after the first pass (which is just the classical ratio test applied to the original problem data), in which case the blocking variable reaches its bound exactly. If $\alpha 1$ is less than the minimum value $\alpha_{\min}$, the blocking variable becomes nonbasic at an infeasible value ($l_r - \delta$ or $u_r + \delta$).

A possible disadvantage of SIMEXPAND is that the pivot element $|p_r|$ is not maximized within a set of candidates. Nevertheless, the final step always satisfies $\alpha \geq \tau/|p_r|$, which tends to prevent selection of a small pivot element unless the feasible region is unbounded. Numerical instability seems unlikely if δ_f and τ have the recommended values, and no numerical difficulties were encountered in the computational tests.

6. Relationship to Wolfe's procedure

Either form of the EXPAND procedure may be interpreted as a modification of Wolfe's "ad hoc" anti-cycling procedure [39].

6.1. *Wolfe's procedure*

We first consider an LP with general lower bounds:

$$\begin{array}{lll} \mathrm{LP}_0 & \text{minimize} & c^{\mathrm{T}}x \\ & \text{subject to} & Ax = b, \quad x \geq l, \end{array}$$

and we assume that the current iterate is feasible.

Wolfe's procedure takes effect when the simplex method encounters a degenerate feasible vertex, denoted by x_0. Once degeneracy has been detected, the basic variables are divided into two categories that depend on x_0: *degenerate* variables that are currently on a bound; and *nondegenerate* variables that are strictly feasible with respect to their bounds. These categories are used to define the following subsidiary linear program:

$$\begin{array}{lll} \mathrm{LP}_1 & \text{minimize} & c^{\mathrm{T}}x \\ & \text{subject to} & Ax = b, \quad x_{\mathrm{D}} \geq l_{\mathrm{D}} - d, \quad x_{\mathrm{N}} \geq l_{\mathrm{N}}, \end{array}$$

where d is a *positive* vector, D denotes the *degenerate variables*, and N denotes the usual *nonbasic* variables. Let $\mathcal{N}$ denote the set of $n-m$ indices of variables in x_{N} at x_0, and $\mathcal{D}$ the set of n_{D} indices of variables in x_{D}. The value n_{D} is called the *degree of degeneracy* of x_0.

Problem LP_1 is the same as LP_0 except for the bounds on the basic variables, which have been relaxed for degenerate variables and removed for nondegenerate variables. Wolfe's procedure was originally described in terms of changes to b rather

than l_D; see [39, 36]. Throughout this discussion we follow Osborne [34, p. 87] in moving bounds away from degenerate variables, rather than vice versa. The concepts are essentially the same, except when both upper and lower bounds are present; see Section 6.4.

Clearly, x_0 is a nondegenerate feasible vertex for LP_1, since exactly $n-m$ variables (x_N) are active at their bounds. When the simplex method is applied to LP_1, the values obtained for x are not directly relevant to LP_0, but the bases generated (and the associated dual variables) have meaning for both problems. Three situations may arise:

(1) A finite optimum is obtained for LP_1, which shows that x_0 is an optimal (albeit degenerate) solution of LP_0. The optimal basis and dual variables for LP_1 correspond to an optimal basis and multipliers for LP_0.

(2) LP_1 is found to be unbounded when a certain nonbasic variable is considered for entry into the basis. The same basis and nonbasic variable produce a *feasible descent direction* for LP_0, which allows movement away from the non-optimal point x_0. (This is the *direction of recession* described by Osborne [34].) Solution of LP_0 can continue from x_0.

(3) A degenerate vertex x_1 for LP_1 arises. A crucial feature of x_1 is that its degree of degeneracy must be *less than* n_D, since at least one variable originally in x_N must have moved away from its bound. We may thus define a subsidiary linear program LP_2 at x_1.

The procedure may be applied recursively whenever a degenerate vertex is encountered. Starting with $k=0$, problem LP_k reaches case 1, 2 or 3 in a finite number of iterations (since the objective function for LP_k decreases montonically). Case 3 leads to a new problem LP_{k+1} but can occur only a finite number of times (since the degree of degeneracy is monotonically decreasing). Eventually, the degeneracy at x_0 is "resolved" by either verifying optimality or moving away from x_0.

Wolfe's procedure is appealing for at least two reasons: it uses the simplex method itself to resolve degeneracy, and it can be implemented with a minimum of overhead (at least for the case $l=0$, $u=\infty$), as shown by Ryan and Osborne [36]. However, it is not without drawbacks. Although a degenerate vertex is unlikely to be encountered in LP_k for $k>0$, particularly if d is defined using random positive numbers, it remains necessary (and inconvenient) to cope with the possibility. Furthermore, a practical Wolfe-based procedure must define "numerical" degeneracy and choose the precise set of degenerate variables. Selecting the "wrong" variables for x_D may lead to inefficiency. For example, suppose that some (even all) of the basic variables are not quite on their bounds and hence are not included in x_D. Only a very short step is likely to be taken after the degeneracy has been "resolved" before the procedure must be invoked again.

6.2. *Parameterization of the subsidiary linear program*

Given the Wolfe subproblem LP_1, consider the effect of introducing a positive parameter γ that multiplies the perturbation d in the bounds on x_D, where we

assume that $\|d\| \approx 1$. The subsidiary problem becomes

$$\begin{aligned} \mathrm{LP}_1(\gamma) \quad & \text{minimize} \quad c^{\mathrm{T}}x \\ & \text{subject to} \quad Ax = b, \quad x_{\mathrm{D}} \geq l_{\mathrm{D}} - \gamma d, \quad x_{\mathrm{N}} \geq l_{\mathrm{N}}. \end{aligned}$$

We now show that the value of γ does not affect the sequence of bases generated in solving $\mathrm{LP}_1(\gamma)$.

Lemma 1. *Suppose that the simplex method is applied to* $\mathrm{LP}_1(\gamma)$, *starting at* x_0 *and using the "textbook" ratio test and the starting basis from* LP_0. *With exact arithmetic, the sequence of bases generated is the same for all positive values of* γ.

Proof. At the first iteration of the simplex method, the search direction p is independent of γ. The only variables in the first ratio test are those in $\mathscr{D}$. For $j \in \mathscr{D}$, the initial value of x_j is l_j and its lower bound in $\mathrm{LP}_1(\gamma)$ is $l_j - \gamma d_j$. The distance of each possible blocking variable x_j from its bound is thus γd_j. Assuming that a blocking variable exists and that a consistent rule is applied for breaking ties, the *same* blocking variable r will be chosen at the first iteration of $\mathrm{LP}_1(\gamma)$ for every positive γ, and the associated steplength satisfies

$$\alpha = \alpha_\gamma = \gamma\alpha^*, \tag{6.1}$$

where α^* is the step at the first iteration of $\mathrm{LP}_1(1)$. The first basis change is independent of γ because the distance of each potential blocking variable from its bound in $\mathrm{LP}_1(\gamma)$ is a factor of γ times the distance from its bound in $\mathrm{LP}_1(1)$.

At the end of the first iteration, each variable x_j, $j \in \mathscr{D}$, takes on the value $l_j + \alpha_\gamma p_j$. We conclude from (6.1) that its distance from the bound $l_j - \gamma d_j$ is

$$l_j + \alpha_\gamma p_j - (l_j - \gamma d_j) = \gamma d_j + \alpha_\gamma p_j = \gamma(d_j + \alpha^* p_j).$$

The newly basic variable (say x_k, $k \in \mathscr{N}$) moves a distance $\alpha_\gamma p_k$ from its bound.

Thus, even after the first basis change, the distance of every possible blocking variable in $\mathrm{LP}_1(\gamma)$ from its bound is γ times the distance from its bound in $\mathrm{LP}_1(1)$. It follows that changes of basis will continue to be independent of γ as long as blocking variables exist. □

For our purposes, the result of Lemma 1 is significant because the Wolfe subproblems may be posed with an *extremely small* value of γ, which implies that there is no need to define and solve LP_1. A "modification" of Wolfe's procedure would simply continue simplex iterations on (conceptually) the original problem LP_0, but with a *tiny* modification to the bounds on the "degenerate" variables x_j, $j \in \mathscr{D}$.

Whenever a degenerate vertex is encountered, another small perturbation with a different vector d would be introduced to the bounds on a new set of degenerate variables. Strictly speaking, this approach would eventually solve a *perturbed* version of the original problem LP_0.

To implement such a modified Wolfe technique, the perturbation vector(s) d need to be defined. Choosing the elements of d at random (in the range $(0, 1]$, say) reduces the probability of creating a further degenerate vertex. From the perspective of preserving well-conditioned bases, however, the "best" choice would be $d = e$, the vector of ones (assuming that the problem is well scaled), since the first blocking variable would correspond to the largest eligible pivot element in the search direction p. Unfortunately, such a structured choice for d would seem to increase the probability of encountering a degenerate vertex. For the original Wolfe procedure, a compromise is to choose elements of d randomly from a range such as $(0.5, 1]$ (cf. [9]), thereby reducing the frequency of recursion beyond LP_1 [36]. However, for the modified procedure, the need to make frequent perturbations is of no consequence provided the total perturbation remains small.

6.3. *Connection with the EXPAND procedure*

The EXPAND procedure may be interpreted as a further modification of the Wolfe procedure, with two key differences:

- a small perturbation in the bounds occurs at *every* iteration regardless of whether or not the current iterate is degenerate;
- a similar perturbation is made to *all* the bounds, not just those for variables that have been declared "degenerate".

The crucial issue in relating EXPAND to Wolfe's method is the effect of the "extra" perturbations in bounds on nondegenerate basic variables. Since the bounds on nondegenerate basic variables are removed in Wolfe's formulation, only the degenerate and nonbasic variables at x_k may be blocking in LP_{k+1}. Although the EXPAND procedure cannot guarantee the latter property, it nonetheless tends to hold because only the variables in x_D and the nonbasic variables are "close to" their bounds at or near a degenerate vertex. Except in extreme cases, a nondegenerate basic variable is unlikely to be chosen as the blocking variable, since its perturbed bounds remain relatively "far away".

The notion of temporary bounds (Section 2.2) can be utilized so that nonbasic variables do not need to be adjusted as their bounds are perturbed. Furthermore, basic variables that subsequently become nonbasic may be set at their original (rather than perturbed) bounds whenever a significant step is taken, to reduce the disturbance caused by occasional resets.

6.4. *Wolfe's procedure with upper and lower bounds*

Consider applying Wolfe's procedure to define LP_1 at a degenerate vertex for an original LP containing both upper and lower bounds. Our view is that the essential ideas in Wolfe's approach are to *remove* inactive constraints, and to relax (by a finite amount) constraints that are active but are not in the working set [21]. Using these guidelines, the subsidiary problem LP_1 would be constructed as follows:

(1) Nonbasic variables currently on a bound would have the opposite bound removed (except for fixed variables, which remain fixed).

(2) Nonbasic variables on neither bound would have both bounds removed.

(3) Degenerate basic variables would have the active bound relaxed. The opposite bound would be removed (except for degenerate fixed variables, which would have both bounds relaxed).

(4) Nondegenerate basic variables would have both bounds removed.

A parameterized subproblem $LP_1(\gamma)$ can then be defined, Lemma 1 still holds, and the EXPAND procedure may be interpreted as a modification of Wolfe's procedure.

An alternative interpretation of a Wolfe procedure for an LP with upper and lower bounds is that nonbasic variables should retain *both* their bounds in the subsidiary linear program LP_1. In this case, the sequence of basis changes in $LP_1(\gamma)$ remains invariant only for values of γ small enough so that the inactive bounds on nonbasic variables at x_0 do not affect the choice of blocking variable. Because the EXPAND procedure corresponds to small values of γ, the spirit of Wolfe's approach is maintained in either case.

As mentioned in Section 6.1, the original Wolfe procedure defines LP_1 by altering the right-hand side b of the equality constraints, and obtains an initial nondegenerate solution by moving components x_j, $j \in \mathcal{D}$, away from their bound. This approach cannot be used with general bounds if there are likely to be fixed basic variables. Any such variable is doubly degenerate (except perhaps in Phase 1) and cannot be moved away from both of its bounds without becoming infeasible.

6.5. Summary

Compared to Wolfe's method, the EXPAND procedure has the following properties:

- there is no need to decide whether or not degeneracy is present, or to specify a set of degenerate variables;
- essentially no storage or logical overhead is involved, and upper and lower bounds can be handled without complication;
- no numerical or logical information need be preserved across basis factorizations (other than the values of x and the current feasibility tolerance);
- all iterations are "equal", in the sense that there is exactly one steplength determination per iteration. (In Wolfe's method, if LP_k is found to be degenerate, the steplength procedure is effectively repeated at the beginning and end of LP_{k+1}.)

An apparent disadvantage is the need to store numerical values for nonbasic variables that are slightly outside their true bound—the simplest approach being to store all of x. The cost is most obvious on problems involving many variables ($n \gg m$, as in crew scheduling problems [10, 9]). However, storing all of x is a convenience for several reasons (Section 2.2) and is essential for implementing the Harris steplength procedure correctly (Section 3.3). Once this overhead is accepted, the ability to implement the EXPAND procedure comes at no further cost.

Several pivot-selection methods have recently been implemented and compared by Falkner [9] on a wide range of aircrew rostering problems involving many variables (21 000 up to 162 000). The qualitative similarity in performance of Wolfe's method and the EXPAND procedure provides empirical confirmation of the relationship described above.

7. Issues arising in Phase 1

Broadly speaking, Phase 1 of the simplex method finds a feasible point for the original constraints $Ax = b$, $l \leq x \leq u$, by applying a normal (Phase 2) procedure to a *modified problem.* The main points of Phase 1 are summarized in Section 7.1, followed by a discussion of some finer points concerning the EXPAND procedure. (See also Wolfe [40], Orchard-Hays [32] and Beale [2].)

7.1. The Phase-1 bounds and objective

The equality constraints $Ax = b$ are retained in the Phase-1 LP, but the *bounds* are altered so that the current point is feasible. (Infeasibilities with respect to the original bounds are reflected in the Phase-1 objective function.) Let δ denote a feasibility tolerance. The Phase-1 bounds $\tilde{l}$ and $\tilde{u}$ are constructed as follows, depending on the bounds violated by the current point x.

if $x_j < l_j - \delta$ **then** $\tilde{l}_j \leftarrow -\infty$; $\tilde{u}_j \leftarrow u_j$
else if $x_j > u_j + \delta$ **then** $\tilde{l}_j \leftarrow l_j$; $\tilde{u}_j \leftarrow +\infty$
else $\tilde{l}_j \leftarrow l_j$; $\tilde{u}_j \leftarrow u_j$ **end if**

Let $\mathcal{J}_u(x)$ and $\mathcal{J}_l(x)$ be the sets of indices of components of x that are infeasible with respect to their upper and lower bounds, respectively:

$$j \in \mathcal{J}_u \quad \text{if } x_j > u_j + \delta, \qquad j \in \mathcal{J}_l \quad \text{if } x_j < l_j - \delta. \tag{7.1}$$

The Phase-1 objective function is the *sum of infeasibilities*:

$$\text{sum of infeasibilities} = \sum_{j \in \mathcal{J}_u} (x_j - u_j - \delta) + \sum_{j \in \mathcal{J}_l} (l_j - \delta - x_j). \tag{7.2}$$

Omitting constants, we define the Phase-1 objective function as $\tilde{c}^T x$, where from (7.2) we see that $\tilde{c}_j = 1$ if $j \in \mathcal{J}_u$, -1 if $j \in \mathcal{J}_l$, and 0 otherwise. Since the indices in $\mathcal{J}_u$ and $\mathcal{J}_l$ vary with x, $\tilde{c}$ may change at every Phase-1 iteration. The current value of $\tilde{c}$ is used to compute reduced costs and a search direction p such that $\tilde{c}^T p < 0$.

Given p, the EXPAND procedure may be applied in Phase 1 to compute a step α_F and blocking variable r_F:

$$(\alpha_F, r_F) = \text{EXPAND}(x, p, \tilde{l}, \tilde{u}, t, \delta, \tau), \tag{7.3}$$

where $\tilde{l}$ and $\tilde{u}$ are the Phase-1 bounds. The step α_F is the largest positive step that *retains feasibility with respect to bounds that are already satisfied.* The value of α is α_F or a special step α_1 (see Section 7.3), subject to safeguards discussed below. Phase 1 is essentially the same as Phase 2 except that $\tilde{c}$, $\tilde{l}$ and $\tilde{u}$ are redefined every iteration, and two possible steps are computed rather than one.

Since p always satisfies $\tilde{c}^T p < 0$, the sum of infeasibilities must (locally) strictly decrease as the step along p increases from zero. When the step becomes large enough so that the index sets $\mathcal{J}_u$ or $\mathcal{J}_l$ change, $\tilde{c}$ also changes, and the sum of infeasibilities may decrease at a lower rate or could even start to increase. However, the *number* of infeasibilities (i.e., the sum of the numbers of indices in $\mathcal{J}_l$ and $\mathcal{J}_u$) will be smaller. Assuming that a positive step is taken at each iteration, we thus see that Phase 1 must converge to a feasible point (if one exists).

A more intricate Phase-1 steplength procedure can be designed to minimize the piece-wise linear function $\tilde{c}^T(x+\alpha p)$, where $\tilde{c}$ is regarded as a function of α; for example, see [23, 15]. However, we adopt the simpler approach, which seems to be effective in practice. Both approaches have the desirable property that many infeasibilities can be removed in one iteration.

7.2. *Benefits of increasing the feasibility tolerance*

We have observed previously that increasing the feasibility tolerance at every iteration ensures that a positive step can be taken. Further, the feasibility tolerance actually affects the sum of infeasibilities. If the number of infeasibilities does not decrease during a particular iteration, the sum of infeasibilities at the start of the next iteration must be reduced simply because the value of δ has increased by τ (see (7.2)). Thus *for two separate reasons*, the sum and/or the number of infeasibilities must decrease after each Phase-1 iteration.

Several complex issues involving finite-precision computation arise in any "practical" anti-cycling method. For example, Fletcher's method for resolving degeneracy is designed to display favorable properties in the presence of rounding error; see [12, 13]. From this perspective, the value of τ in EXPAND can be viewed as a means of coping with certain numerical difficulties. Although τ is typically very small, it is intended to be significantly larger than machine precision ε, and preferably larger than the tolerance t that defines "ignored" components of p (Sections 3.1 and 4.5). Increasing δ and τ thus *helps mask the rounding error* that is inevitably present when x is updated to $x+\alpha p$, and guarantees that the number of infeasible variables (as measured by the increased tolerance) will stay the same or decrease. We believe that many "infinite loop" failures of simplex implementations are attributable to an inadvertent *oscillation* in the number of infeasibilities when $\tilde{c}$ is redefined each Phase-1 iteration with a fixed feasibility tolerance. An example is described by Ogryczak [31]. Similar examples were encountered with MINOS prior to the present implementation.

After K iterations are executed in Phase 1, the feasibility tolerance is reduced to δ_0 for the next expanding sequence. An apparent disadvantage is that the number of infeasibilities may increase by some arbitrary number (say q), and the sum could increase by as much as $q(\delta_K - \delta_0)$. However, this fluctuation is normally inconsequential even if q is nearly as large as m, primarily because many infeasibilities tend to be removed in a single Phase-1 iteration. Similar comments apply when the resetting procedure is invoked at an apparently optimal solution.

7.3. The special Phase-1 step

The Phase-1 objective (7.2) is constructed so that at least some of the infeasible variables move towards the feasible region. The "alternative" Phase-1 step α_1 is based on finding the positive step at which these variables first become feasible. Let $\mathcal{J}$ denote the (necessarily non-empty) set of indices with the following properties:

$$\mathcal{J} = \{j \mid j \in \mathcal{J}_u \text{ and } p_j < 0 \text{ or } j \in \mathcal{J}_l \text{ and } p_j > 0\}.$$

The function step_feas for which pseudo-code is given below is a Phase-1 analogue of the function step of Section 3.1. For indices in $\mathcal{J}$, step_feas gives the (positive) multiple of ρ that reaches the nearer bound.

function step_feas(χ, ρ, l, ν, t);
if $\rho > t$ **then** step_feas $\leftarrow (l - \chi)/\rho$
  else if $\rho < -t$ **then** step_feas $\leftarrow (\nu - \chi)/\rho$
  else step_feas $\leftarrow -\infty$
end if

When the tolerance t is omitted from the parameters of step_feas, it should be taken as zero.

The special step α_1 corresponds to the value of step_feas(x_s, p_s, l_s, u_s) for some index $s \in \mathcal{J}$, and we now consider how to choose s. Let $\tilde{\alpha}_{\max}$ denote the maximum value of step_feas:

$$\tilde{\alpha}_{\max} = \max_{j \in \mathcal{J}} \text{step\_feas}(x_j, p_j, l_j, u_j),$$

which must be positive because $\mathcal{J}$ is non-empty. If any infeasible variables become feasible as the step increases, $\tilde{\alpha}_{\max}$ gives the step at which the largest number become feasible *with respect to their nearer bound.* Note, however, that a step of $\tilde{\alpha}_{\max}$ can cause some of the variables to violate a bound that they previously satisfied.

An obvious strategy is to define $\alpha_1 = \tilde{\alpha}_{\max}$ and $\alpha = \min\{\alpha_1, \alpha_F\}$. By requiring α to be less than α_F, we guarantee that variables remain feasible with respect to currently satisfied bounds. The upper bound of α_1 is imposed to ensure that a finite step is taken when α_F is infinite (which may occur when some components of l or u are infinite).

A difficulty with letting $\alpha_1 = \tilde{\alpha}_{\max}$ is that no account is taken of the size of the pivot element. Following the philosophy of Harris (Section 3.2), the following two-pass procedure may be used to encourage selection of larger pivot elements when a choice exists. Placing perturbations on the bounds when calling step_feas in the first pass makes $\tilde{\alpha}1$ *smaller* than the maximum step that causes the variable to become feasible. The second pass then considers all unperturbed values of step_feas that are at least as large as $\tilde{\alpha}1$, and chooses s to correspond to the maximal pivot element among them.

$\tilde{\alpha}1 = \max_{j \in \mathcal{J}}$ step_feas(x_j, p_j, $l_j - \delta$, $u_j + \delta$, t); (first pass)
$s \leftarrow 0$; $p_{\max} \leftarrow 0$;
for $j \in \mathcal{J}$ **do** (second pass)

$\tilde{\alpha}_j \leftarrow \text{step\_feas}(x_j, p_j, l_j, u_j, t)$;
if $\tilde{\alpha}_j \geq \tilde{\alpha} 1$ **and** $|p_j| > p_{\max}$ **then**
$s \leftarrow j$; $\alpha_1 \leftarrow \tilde{\alpha}_j$; $p_{\max} \leftarrow |p_j|$
end if
end for

The above procedure was used in Lustig's experiments [26] and in all preceding versions of MINOS. It appears to have performed reliably for many years.

An undesirable feature is that the chosen pivot $|p_s|$ could still be as small as the tolerance t. The safer two-pass strategy given below has therefore been adopted. The first pass finds the largest relevant pivot element ϕ, and the second pass finds the largest value of step_feas for which the pivot element is "reasonably close" to ϕ for some constant ω, where $0 < \omega \leq 1$.

$\phi \leftarrow \max_{j \in \mathcal{J}} |p_j|$; (first pass)
$\alpha_I \leftarrow 0$;
for $j \in \mathcal{J}$ **do** (second pass)
$\tilde{\alpha}_j \leftarrow \text{step\_feas}(x_j, p_j, l_j, u_j, t)$;
if $\tilde{\alpha}_j > \alpha_I$ **and** $|p_j| > \omega\phi$ **then** $s \leftarrow j$; $\alpha_1 \leftarrow \tilde{\alpha}_j$ **end if**
end for

Computation of α_1 does not depend critically on the feasibility tolerance, and is hence compatible with the EXPAND procedure.

Experience suggests that the step α_1 should be taken whenever possible (in preference to α_F), so that x_s reaches a bound and is removed from the basis. We therefore define the Phase-1 step as

$$\alpha = \begin{cases} \alpha_1, & \alpha_1 \leq \alpha 1, \\ \alpha_F, & \text{otherwise.} \end{cases}$$

The value of $\alpha 1$ comes from the first pass of the Harris procedure that computes α_F, and is therefore larger than α_F.

On the 53 test problems of Section 9, the values $\omega = 0.1$ and $\omega = 0.01$ lead to more Phase-1 iterations than the unsafeguarded $\omega = 0$. We have accordingly chosen $\omega = 0.001$.

8. Nonlinear programs with linear constraints

We now consider the problem of minimizing a smooth function $F(x)$ subject to linear constraints. This category includes quadratic programs (QP) and more general linearly constrained (LC) optimization problems.

It has been observed by Osborne [34] that Wolfe's anti-cyling procedure generalizes to certain LC algorithms, including the reduced-gradient method of Wolfe [38]. The EXPAND procedure can similarly be generalized to active-set methods for QP and LC problems (Fletcher [11]; Gill, Murray and Wright [21]), and has been

implemented in the 1988 versions of QPSOL and LSSOL. The following preliminary strategy has been developed for the reduced-gradient algorithm in MINOS.

8.1. A normal iteration

In an active-set framework, the EXPAND procedure may be applied directly to a general linearly constrained problem. Each iteration has the generic form (2.2): $x \leftarrow x + \alpha p$, where p is a search direction such that $\nabla F(x)^{\mathrm{T}} p < 0$ and α is a nonnegative steplength. After p is computed, a positive step and *blocking index* are computed. The step is traditionally called the "step to the nearest constraint", and will be denoted by α_{blk}. Any value of α in the interval $(0, \alpha_{\mathrm{blk}}]$ will produce a new iterate that is acceptably feasible.

With a linear program, the objective function is monotonically decreasing along p, and α is limited only by considerations of feasibility. With a nonlinear objective function, however, α is based on two considerations: maintaining feasibility *and* achieving the classical "sufficient decrease" in F (see [33]). The latter is often based on approximate minimization of F as α ranges over the interval $(0, \alpha_{\mathrm{blk}}]$. With the EXPAND procedure,

$$\alpha_{\mathrm{blk}} = \max\{\alpha_{\min}, \alpha 2\},$$

where $\alpha 2$ is the exact step to the blocking constraint.

If a sufficiently large step can be taken, the objective function is strictly reduced and there is no danger of cycling. If $\alpha = \alpha_{\mathrm{blk}}$, the blocking constraint is added to the working set.

8.2. Avoiding the linesearch

In practice, it may be inefficient or unwise to attempt a linesearch when α_{blk} is very small (for example, if the active constraints are almost linearly dependent). Even if the maximum step α_{blk} is taken, the improvement in objective value may be slight. More seriously, the "noise level" in F over the permitted interval $(0, \alpha_{\mathrm{blk}}]$ may be so great that an improved point cannot be conclusively identified, and the linesearch will "fail".

To avoid these situations, we use the step $\alpha 2$ (which may be negative) to the blocking constraint. If $\alpha 2 > 0$, a linesearch is always performed. If $\alpha 2 \leq 0$, the active constraints are "nearly" linearly dependent, and a *zero step* is usually taken by skipping the linesearch and adding the blocking constraint to the working set. The only exception to the latter policy occurs when adding the blocking constraint would create a vertex of the feasible region, since this circumstance combined with a zero step could lead to cycling. If $\alpha 2 \leq 0$ and adding the blocking constraint to the working set would create a vertex, a linesearch is performed over the interval $(0, \alpha_{\mathrm{blk}}]$.

If the linesearch ever fails to find an improved point, an effort is made to determine whether the failure was caused by too small a search interval. If $\alpha 2 < \alpha_{\min}$, a step to the blocking constraint is forced ($\alpha = \alpha 2$) and the working set is updated. Otherwise, we assume that a better search direction is required. The working set is

left unaltered and various recovery procedures are invoked, such as switching to central-difference gradient approximations, resetting the reduced Hessian approximation, deleting a constraint from the working set, and refactorizing the working set.

Because many methods for problems with nonlinear constraints are based on solving a sequence of linearly constrained subproblems (e.g., SQP methods and MINOS), the EXPAND procedure may be applied within the subproblems.

9. Computational results

This section contains computational results for three steplength procedures. The following names are used:

SP1: The "textbook" ratio test of Section 3.2.

SP2: The simplified EXPAND procedure of Section 5.

SP3: The maximum-pivot EXPAND procedure of Section 4, which includes Harris-type tie-breaking.

All three procedures have been implemented in MINOS 5.3 (June 1989); SP3 has also been implemented in GAMS/MINOS [4].

One aim is to provide a systematic study of the effect of maximizing the pivot element within a steplength procedure—the Harris approach to tie-breaking. Folklore has it that "stability is improved and the number of simplex iterations is often reduced", but such a statement is not especially meaningful without a precise definition of the procedures being compared. Here we have defined the procedures in appropriate detail. In particular, it is meaningful to compare the simplified and standard EXPAND procedures because in both cases the surrounding simplex algorithm retains the correct numerical values of blocking variables when they become nonbasic.

The results below were obtained using the simplex method of MINOS 5.3 on the first 53 linear programs in the *Netlib* collection [17]. The problems are ordered according to the number of nonzero elements as in [26]. The main run-time options specified were

PRINT LEVEL	0
CRASH OPTION	1
CRASH TOLERANCE	0.1
SCALE OPTION	2
PARTIAL PRICE	10
LU FACTOR TOLERANCE	100.0
FACTOR FREQUENCY	100
EXPAND FREQUENCY	10000
FEASIBILITY TOLERANCE	1.0 E −6

which are the default options for linear problems in MINOS 5.3. The last two options define $K = 10\,000$ and $\delta_f = 10^{-6}$ for the EXPAND procedures. The limit on calls to the resetting procedure after apparent termination was set to $R = 2$ (Section 4.3).

The CRASH parameters above cause MINOS to choose an approximately triangular basis from the columns of A. In most cases the chosen scaling option has the effect of making $\|\tilde{x}^*\| = O(1)$, where $\tilde{x}^*$ is the scaled optimal solution. (Exceptions were problems GROW7, GROW15 and GROW22, for which $\|\tilde{x}^*\| = O(10^7)$, $\|x^*\| = O(10^6)$.) Selection of this scaling option helps to justify the choice of $\delta_f = 10^{-6}$ as a feasibility tolerance.

All numerical tests were run as batch jobs on a DEC VAX station II with the VAX/VMS version 4.5 operating system. The compiler was VAX FORTRAN version 4.6 with default options, including code optimization and D_floating arithmetic (relative precision $\varepsilon \approx 2.8 \times 10^{-17}$). The memory available kept paging to a minimum.

Tables 1, 2 and 3 give results using SP1, SP2 and SP3 respectively. The "objective function" values indicate that the final objective was accurate to four or more digits (except for two problems that terminated early). The meaning of "degenerate steps" depends on the method, as discussed below. Solution times are given in CPU seconds, and do not include time for data input or solution output.

9.1. *The textbook ratio test*

SP1 was safeguarded by choosing the tolerance t that defines "negligible" components of p as $\varepsilon^{2/3}$ (see Sections 3.1 and 4.5). Since rounding error can cause the steplength to be negative, a further precaution was to set $\alpha = 0$ if ratio_test gave $\alpha \leq 10^{-16}$. (The count of "degenerate steps" in Table 1 gives the number of times α was set to zero in this fashion.) Following conventional practice, blocking variables were set exactly on their bounds when they became nonbasic.

Although it would be reasonably easy to break (near) ties in favor of large $|p_j|$, we chose not to tamper further with the classical procedure; methodical tie-breaking is the province of the Harris and EXPAND procedures.

In the test runs, small pivots slipped through the $\varepsilon^{2/3}$ sieve several times on each of the problems GROW7, GROW15, GROW22, SCSD1, SCSD8, FFFFF800, PILOTJA and PILOTS. In general, small pivots are detected as near-singularities when the LU factors of the basis are updated. Refactorization is invoked and some variable x_j is replaced by an appropriate slack variable. Since x_j retains its value when rejected from the basis, iterations continue without apparent interruption.

Two failures were encountered: problem SCSD8 terminated as "unbounded", and PILOTS terminated after not changing the sum of infeasibilities for 1000 iterations. Small pivots were encountered frequently during these runs, causing the basis to be ill-conditioned for many groups of iterations. Empirically, ill-conditioning can only aggravate stalling (particularly for a method that has no guarantee of terminating).

9.2. *The simplified EXPAND procedure*

For SP2, the value of "degenerate steps" in Table 2 means the number of times

Table 1

Results with textbook ratio test

Problem (Netlib)	Objective function	Total itns.	Degen. steps	Percent degen.	Solve time VAX II secs.
AFIRO	−4.6475314285714 E +02	9	5	55.6	0.53
ADLITTLE	2.2549496316238 E +05	117	13	11.1	5.94
SC205	−5.2202061211707 E +01	163	48	29.5	17.48
SCAGR7	−2.3313897523795 E +06	98	11	11.2	7.46
SHARE2B	−4.1573224074142 E +02	137	37	27.0	9.42
RECIPE	−2.6661600000000 E +02	27	3	11.1	1.66
VTPBASE	1.2983146246136 E +05	136	99	72.8	11.89
SHARE1B	−7.6589318579186 E +04	197	9	4.6	16.65
BORE3D	1.3730803942085 E +03	160	102	63.8	20.73
SCORPION	1.8781248227381 E +03	172	66	38.4	30.51
CAPRI	2.6900129137682 E +03	246	42	17.1	31.19
SCAGR25	−1.4753433060769 E +07	338	55	16.3	78.92
SCTAP1	1.4122500000000 E +03	346	175	50.6	44.71
BRANDY	1.5185098964881 E +03	472	61	12.9	79.77
ISRAEL	−8.9664482186305 E +05	255	3	1.2	31.89
ETAMACRO	−7.5571521755573 E +02	687	189	27.5	139.20
SCFXM1	1.8416759028349 E +04	442	116	26.2	75.53
GROW7	−4.7787811814712 E +07	683	593	86.8	201.83
BANDM	−1.5862801845012 E +02	475	59	12.4	98.46
E226	−1.8751929066371 E +01	523	167	31.9	73.51
STANDATA	1.2576995000000 E +03	72	44	61.1	12.22
SCSD1	8.6666666743334 E +00	1394	1339	96.0	115.03
GFRDPNC	6.9022359995488 E +06	630	335	53.2	157.03
BEACONFD	3.3592485807200 E +04	91	17	18.7	9.85
STAIR	−2.5126695119296 E +02	628	181	28.8	247.13
SCRS8	9.0429998618888 E +02	788	457	58.0	188.69
SEBA	1.5711600000000 E +04	365	55	15.1	76.42
SHELL	1.2088253460000 E +09	300	73	24.3	70.90
PILOT4	−2.5811392588836 E +03	1651	190	11.5	614.92
SCFXM2	3.6660261564999 E +04	767	186	24.2	238.26
SCSD6	5.0500000078267 E +01	1940	1165	60.0	269.30
GROW15	−1.0687094129358 E +08	1204	1073	89.1	1053.40
SHIP04S	1.7987147004454 E +06	162	35	21.6	33.52
FFFFF800	5.5567956521288 E +05	1234	618	50.1	356.99
GANGES	−1.0958589354318 E +05	791	200	25.3	361.58
SCFXM3	5.4901254549751 E +04	1088	251	23.1	492.69
SCTAP2	1.7248071428571 E +03	883	619	70.1	345.21
GROW22	−1.6083433648256 E +08	1579	1387	87.8	2578.05
SHIP04L	1.7933245379704 E +06	297	73	24.6	63.71
PILOTWE	−2.7200967172270 E +06	5145	961	18.7	2565.39
SIERRA	1.5394362183632 E +07	796	400	50.2	355.81
SHIP08S	1.9200982105346 E +06	236	55	23.3	90.86
SCTAP3	1.4240000000000 E +03	1503	1221	81.2	787.19
SHIP12S	1.4892361344061 E +06	436	117	26.8	231.64
25FV47	5.5018467790995 E +03	7701	889	11.5	4691.38
SCSD8	1.5676484965792 E +03	2930	1411	48.2	unbounded
NESM	1.4076057772814 E +07	3315	5	0.2	1236.99
CZPROB	2.1851966988566 E +06	1724	130	7.5	755.26
PILOTJA	−6.1131349867462 E +03	6935	930	13.4	4640.94
SHIP08L	1.9090552113891 E +06	497	86	17.3	213.63
SHIP12L	1.4701879193293 E +06	961	295	30.7	520.68
80BAU3B	9.8722799393135 E +05	11137	2291	20.6	9999.93
PILOTS	−4.1351068600000 E +02	1226	1225	99.9	stalled

Table 2
Results with simplified EXPAND procedure

Problem (Netlib)	Objective function	Total itns.	Degen. steps	Percent degen.	Solve time VAX II secs.
AFIRO	−4.6475314285714 E +02	9	5	55.6	0.55
ADLITTLE	2.2549496316238 E +05	119	8	6.7	6.31
SC205	−5.2202061211707 E +01	152	33	21.7	18.70
SCAGR7	−2.3313897523795 E +06	98	8	8.2	7.65
SHARE2B	−4.1573224074142 E +02	133	27	20.3	10.25
RECIPE	−2.6661600000000 E +02	27	3	11.1	1.77
VTPBASE	1.2983146246136 E +05	227	98	43.2	22.26
SHARE1B	−7.6589318579185 E +04	244	13	5.3	22.23
BORE3D	1.3730803942085 E +03	164	68	41.5	22.70
SCORPION	1.8781248227381 E +03	175	60	34.3	32.97
CAPRI	2.6900129137682 E +03	233	31	13.3	29.03
SCAGR25	−1.4753433060769 E +07	334	24	7.2	77.63
SCTAP1	1.4122500000000 E +03	374	115	30.8	49.73
BRANDY	1.5185098964881 E +03	387	28	7.2	66.45
ISRAEL	−8.9664482186305 E +05	224	4	1.8	26.73
ETAMACRO	−7.5571521647657 E +02	600	136	22.7	120.27
SCFXM1	1.8416759028349 E +04	337	52	15.4	56.94
GROW7	−4.7787811814712 E +07	213	74	34.7	43.15
BANDM	−1.5862801845006 E +02	413	27	6.5	89.97
E226	−1.8751929066371 E +01	467	67	14.4	69.67
STANDATA	1.2576995000000 E +03	46	18	39.1	9.39
SCSD1	8.6666666743334 E +00	337	274	81.3	31.50
GFRDPNC	6.9022359995488 E +06	658	259	39.4	169.52
BEACONFD	3.3592485807200 E +04	91	14	15.4	10.07
STAIR	−2.5126695119296 E +02	491	66	13.4	203.52
SCRS8	9.0429998618888 E +02	768	251	32.7	201.48
SEBA	1.5711600000000 E +04	350	42	12.0	73.81
SHELL	1.2088253460000 E +09	301	51	16.9	72.39
PILOT4	−2.5811392614137 E +03	1484	138	9.3	560.01
SCFXM2	3.6660261564999 E +04	711	122	17.2	223.78
SCSD6	5.0500000078262 E +01	1162	616	53.0	162.44
GROW15	−1.0687094129358 E +08	443	131	29.6	160.84
SHIP04S	1.7987147004454 E +06	163	26	16.0	34.74
FFFFF800	5.5567957127313 E +05	953	224	23.5	272.83
GANGES	−1.0958591516963 E +05	780	228	29.2	370.06
SCFXM3	5.4901254549751 E +04	1020	157	15.4	460.32
SCTAP2	1.7248071428571 E +03	1109	544	49.0	457.63
GROW22	−1.6083433648256 E +08	664	206	31.0	338.82
SHIP04L	1.7933245379704 E +06	288	45	15.6	63.62
PILOTWE	−2.7201041578556 E +06	5357	536	10.0	2673.63
SIERRA	1.5394362183632 E +07	685	206	30.1	316.62
SHIP08S	1.9200982105346 E +06	258	52	20.2	97.75
SCTAP3	1.4240000000000 E +03	1379	765	55.5	740.52
SHIP12S	1.4892361344061 E +06	467	77	16.5	252.70
25FV47	5.5018467791002 E +03	6682	426	6.4	4074.34
SCSD8	9.0499999992546 E +02	4012	1692	42.2	1401.92
NESM	1.4076055975501 E +07	3231	1	0.0	1214.64
CZPROB	2.1851966988566 E +06	1661	41	2.5	734.23
PILOTJA	−6.1131353307246 E +03	7515	540	7.2	5069.55
SHIP08L	1.9090552113891 E +06	494	61	12.4	213.92
SHIP12L	1.4701879193293 E +06	958	190	19.8	553.11
80BAU3B	9.8722733242195 E +05	11866	1819	15.3	10917.99
PILOTS	−5.5740422249779 E +02	14953	1848	12.4	35210.24

that two passes were required to determine a blocking variable. No singularities were encountered during the tests, and all problems terminated successfully.

On 80BAU3B and PILOTS, the resetting procedure was invoked after $K = 10\,000$ iterations, with 1152 and 393 nonbasic variables respectively being moved onto their bounds. Feasibility was restored 11 and 1 iterations later.

With δ_f as small as 10^{-6}, resets do not disturb x greatly if the basis is reasonably well-conditioned. After resetting at an apparent optimum, most problems were immediately confirmed as optimal.

Five problems did require further iterations. On PILOT4, SCFXM3, PILOTJA, 80BAU3B and PILOTS, 36, 203, 69, 78 and 61 nonbasic variables (respectively) were moved onto their bound, and 4, 1, 3, 15 and 25 additional iterations were performed. A second reset moved 0, 1, 1, 9 and 0 nonbasics, and the middle three problems then required 0, 0 and 4 final iterations.

9.3. The EXPAND procedure

For SP3, "degenerate steps" in Table 3 means the number of times that α was forced to take the value $\alpha_{\min}$ rather than $\alpha 2$, i.e., the number of times a blocking variable was made nonbasic at an infeasible value, rather than reaching its bound exactly.

On 80BAU3B and PILOTS, the reset after 10 000 iterations moved 1291 and 318 nonbasics respectively, and feasibility was restored 3 and 7 iterations later.

Only two problems continued after resetting at an apparent optimum. On PILOTJA and PILOTS, 91 and 33 nonbasics (respectively) were moved onto their bound, and 74 and 69 additional iterations were performed. A second reset moved 5 and 1 nonbasics, and no further iterations were required.

9.4. Comments on the results

Figures 4–8 summarize the highlights of Tables 1–3. Figure 4 shows the ratios of solution times for SP1 and SP3, Figure 5 gives a sorted version of the same information, and Figure 6 gives the ratios of iteration counts. (Problems SCSD8 and PILOTS have been omitted because of the failure of SP1 to reach an optimal solution.) In all figures, the ratios are plotted on a log scale.

It is clear from Figures 4 and 6 that the outliers are the same with either measure, but the time comparison tends to be more dramatic. Figure 5 reveals that for two-thirds of the solved problems (35 of 51), the solution times for SP1 and SP3 vary by less than 15%. The three greatest improvements in speed for SP3 (by factors of more than 5) occurred on the GROW problems. Much of the difference was due to additional refactorizations for SP1, following detection of singularity.

Figure 7 shows the ratio of solution times for SP2 and SP3, again on a log scale, and Figure 8 gives a sorted version of the same data. Not surprisingly, the performance of SP2 and SP3 is much closer than that of SP1 and SP3. Figure 8 reveals that SP2 was more than 15% faster on 5 problems, whereas SP3 was more than 15%

Table 3

Results with EXPAND procedure

Problem (Netlib)	Objective function	Total itns.	Degen. steps	Percent degen.	Solve time VAX II secs.
AFIRO	−4.6475314285714 E +02	9	5	55.6	0.53
ADLITTLE	2.2549496316238 E +05	121	12	9.9	6.49
SC205	−5.2202061211707 E +01	141	39	27.7	17.79
SCAGR7	−2.3313897523795 E +06	98	10	10.2	7.99
SHARE2B	−4.1573224074142 E +02	173	36	20.8	12.85
RECIPE	−2.6661600000000 E +02	27	3	11.1	1.86
VTPBASE	1.2983146246136 E +05	152	50	32.9	15.02
SHARE1B	−7.6589318579186 E +04	266	2	0.8	24.57
BORE3D	1.3730803942085 E +03	144	51	35.4	19.72
SCORPION	1.8781248227381 E +03	178	59	33.2	33.58
CAPRI	2.6900129137682 E +03	271	41	15.1	35.98
SCAGR25	−1.4753433060769 E +07	338	27	8.0	79.36
SCTAP1	1.4122500000000 E +03	264	97	36.7	35.45
BRANDY	1.5185098964881 E +03	369	39	10.6	64.56
ISRAEL	−8.9664482186305 E +05	251	3	1.2	32.97
ETAMACRO	−7.5571521832862 E +02	567	173	30.5	114.08
SCFXM1	1.8416759028349 E +04	375	63	16.8	63.72
GROW7	−4.7787811814712 E +07	184	54	29.4	34.97
BANDM	−1.5862801845012 E +02	457	41	9.0	95.75
E226	−1.8751929066371 E +01	545	92	16.9	78.29
STANDATA	1.2576995000000 E +03	65	36	55.4	12.40
SCSD1	8.6666666743334 E +00	303	169	55.8	27.83
GFRDPNC	6.9022359995488 E +06	672	304	45.2	179.91
BEACONFD	3.3592485807200 E +04	91	14	15.4	10.56
STAIR	−2.5126695119296 E +02	577	70	12.1	249.99
SCRS8	9.0429998618888 E +02	743	215	28.9	193.33
SEBA	1.5711600000000 E +04	351	39	11.1	78.79
SHELL	1.2088253460000 E +09	299	58	19.4	70.09
PILOT4	−2.5811392640909 E +03	1543	149	9.7	595.87
SCFXM2	3.6660261564999 E +04	670	109	16.3	210.92
SCSD6	5.0500000078262 E +01	1306	597	45.7	180.95
GROW15	−1.0687094129358 E +08	425	95	22.4	164.54
SHIP04S	1.7987147004454 E +06	163	26	16.0	33.57
FFFFF800	5.5567959102689 E +05	796	242	30.4	229.97
GANGES	−1.0958636378469 E +05	757	187	24.7	364.23
SCFXM3	5.4901254549751 E +04	1008	164	16.3	462.68
SCTAP2	1.7248071428571 E +03	761	389	51.1	308.59
GROW22	−1.6083433648256 E +08	634	148	23.3	339.95
SHIP04L	1.7933245379704 E +06	291	38	13.1	63.87
PILOTWE	−2.7201044816159 E +06	5458	527	9.7	2784.98
SIERRA	1.5394362183632 E +07	648	236	36.4	295.85
SHIP08S	1.9200982105346 E +06	254	51	20.1	97.42
SCTAP3	1.4240000000000 E +03	904	506	56.0	481.26
SHIP12S	1.4892361344061 E +06	437	95	21.7	231.42
25FV47	5.5018458882868 E +03	6446	652	10.1	4005.84
SCSD8	9.0499999992546 E +02	3138	1285	41.0	1136.22
NESM	1.4076057079146 E +07	3228	40	1.2	1252.46
CZPROB	2.1851966988566 E +06	1694	102	6.0	749.33
PILOTJA	−6.1131581690180 E +03	6487	643	9.9	4506.88
SHIP08L	1.9090552113891 E +06	474	59	12.4	202.78
SHIP12L	1.4701879193293 E +06	959	256	26.7	563.78
80BAU3B	9.8722740952342 E +05	10166	1845	18.2	9184.18
PILOTS	−5.5740380065647 E +02	13723	1459	10.6	32200.90

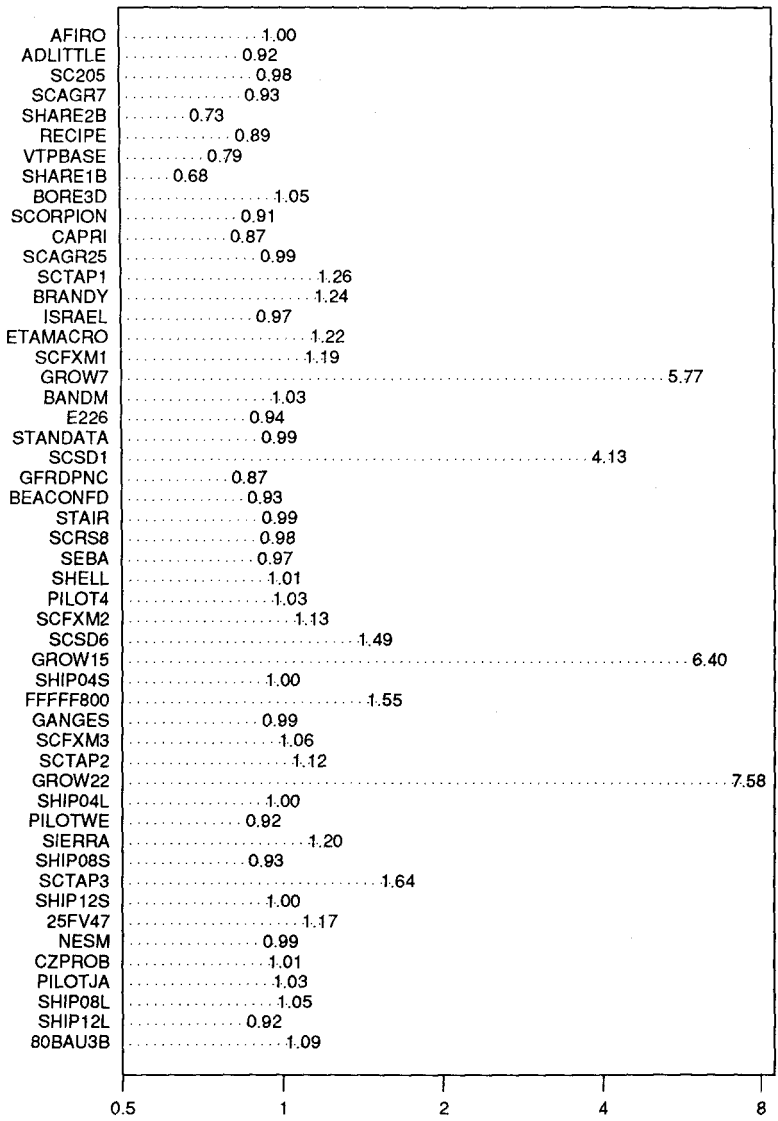

Fig. 4. Comparison of solution times for SP1 and SP3.

faster on 8 problems. (The iteration ratios were qualitatively very similar to the time ratios, and hence are not shown.)

It is interesting to compare the percentage of degenerate iterations for the three strategies. The percentage of degenerate steps with SP1 was more than 15% higher than with SP3 for nearly four-fifths of the problems (with the striking exception of NESM, which had only a tiny percentage of degenerate steps in all cases), and more than twice as large for 9 problems. In contrast, SP2 led to a reduction of 15% or

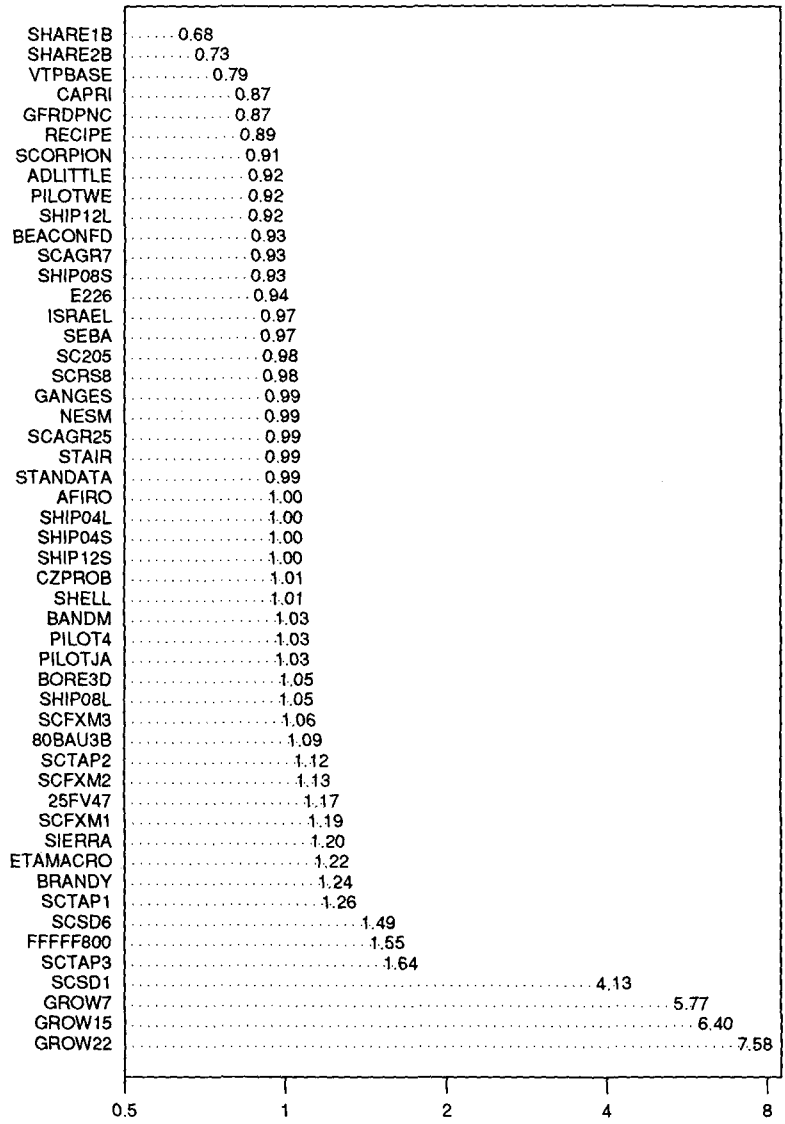

Fig. 5. Sorted comparison of solution times for SP1 and SP3.

more in the percentage of degenerate steps for 10 of the problems, and to an increase of more than 15% for 12 of the problems.

9.5. *Other parameter values*

The 53 test problems have been solved many times, with and without scaling and partial pricing. One of the main parameters of interest is the feasibility tolerance. We have experimented with the values $\delta = 10^{-4}$, 10^{-5}, 10^{-6} and 10^{-7} (Harris recommended $\delta = 5 \times 10^{-4}$ on a machine with $\varepsilon \approx 10^{-8}$), but the sensitivity of the simplex

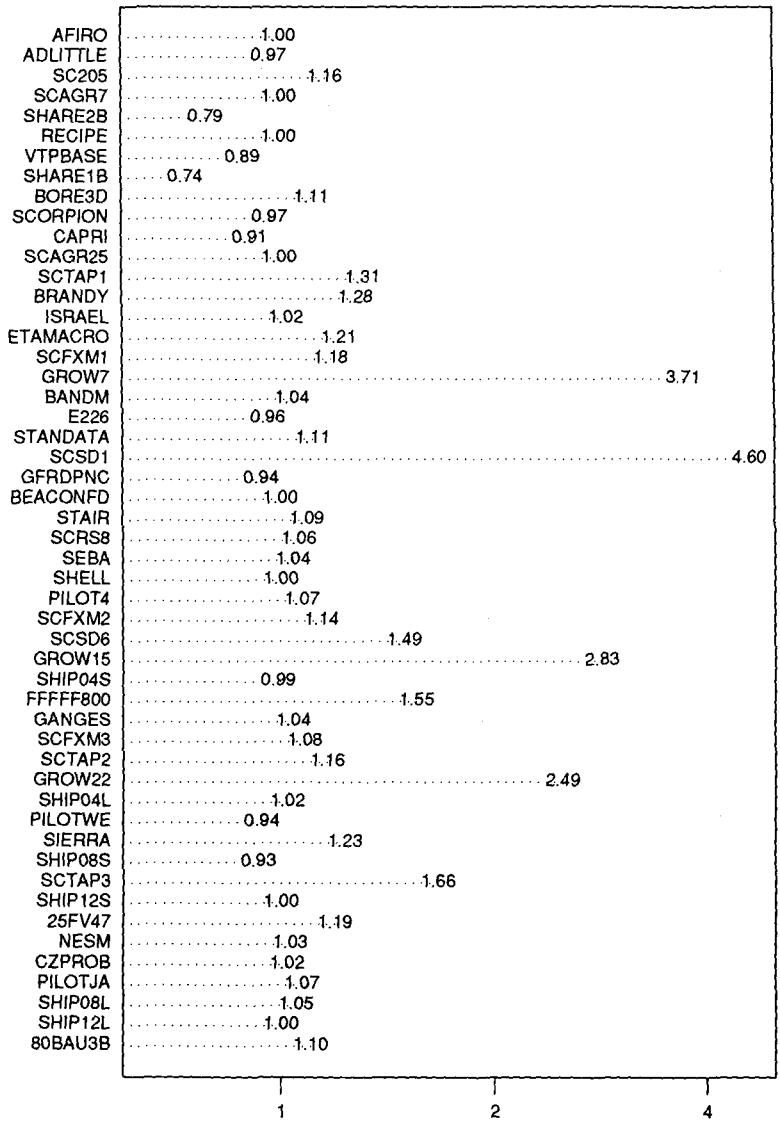

Fig. 6. Comparison of iteration counts for SP1 and SP3.

method to minor algorithmic changes seems to have masked any useful trend. Significant improvements were certainly observed on some of the problems with $\delta = 10^{-4}$. The risk is a greater disturbance after resetting on problems that are somewhat ill-conditioned (notably PILOTJA and PILOTS).

As a further test, the "expand" feature of SP3 was disabled by specifying $K = 99\,999\,999$, $\tau = 0$. The resulting method retains a *fixed* feasibility tolerance, and most closely resembles the Harris tie-breaking procedure. No failures occurred on four

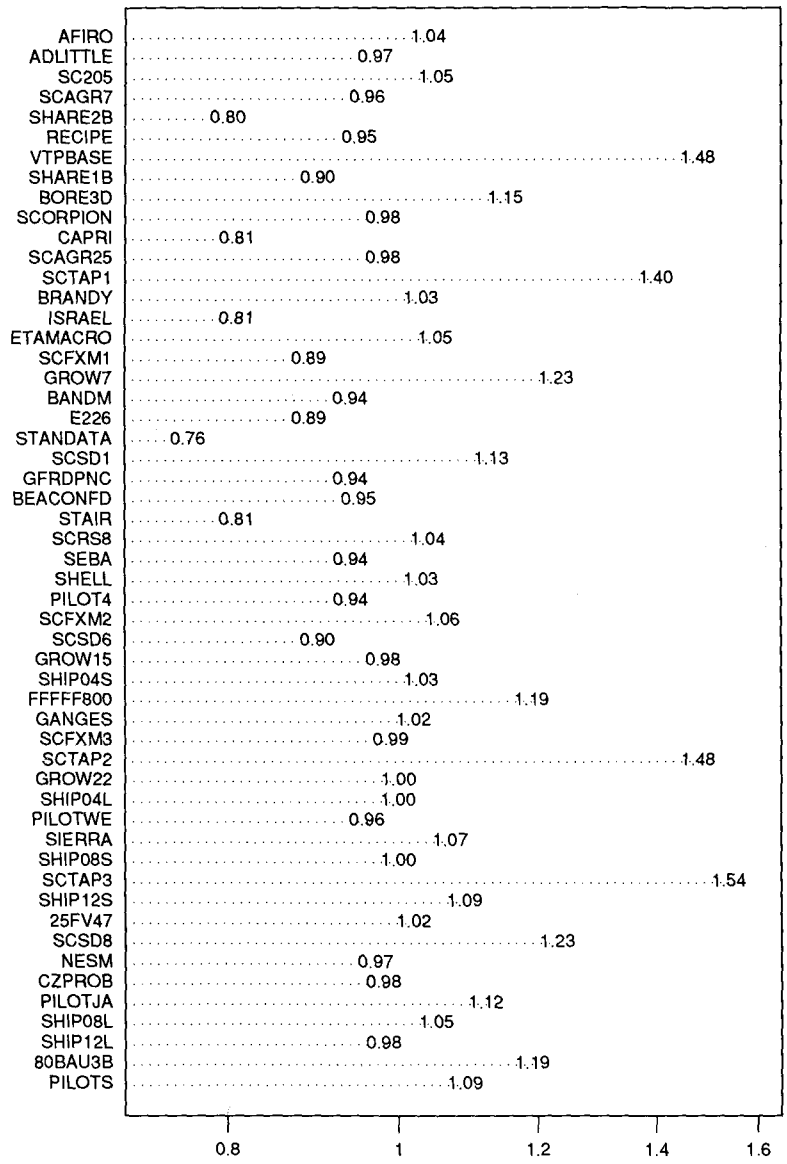

Fig. 7. Comparison of solution times for SP2 and SP3.

runs with and without scaling and partial pricing, and the iteration counts were much the same as when the feasibility tolerance is increased. These results confirm that the probability of failure with the Harris procedure is indeed low when blocking variables are made nonbasic at their correct value. However, once the latter safeguard is implemented, the assurance gained by increasing the feasibility tolerance comes at essentially no cost.

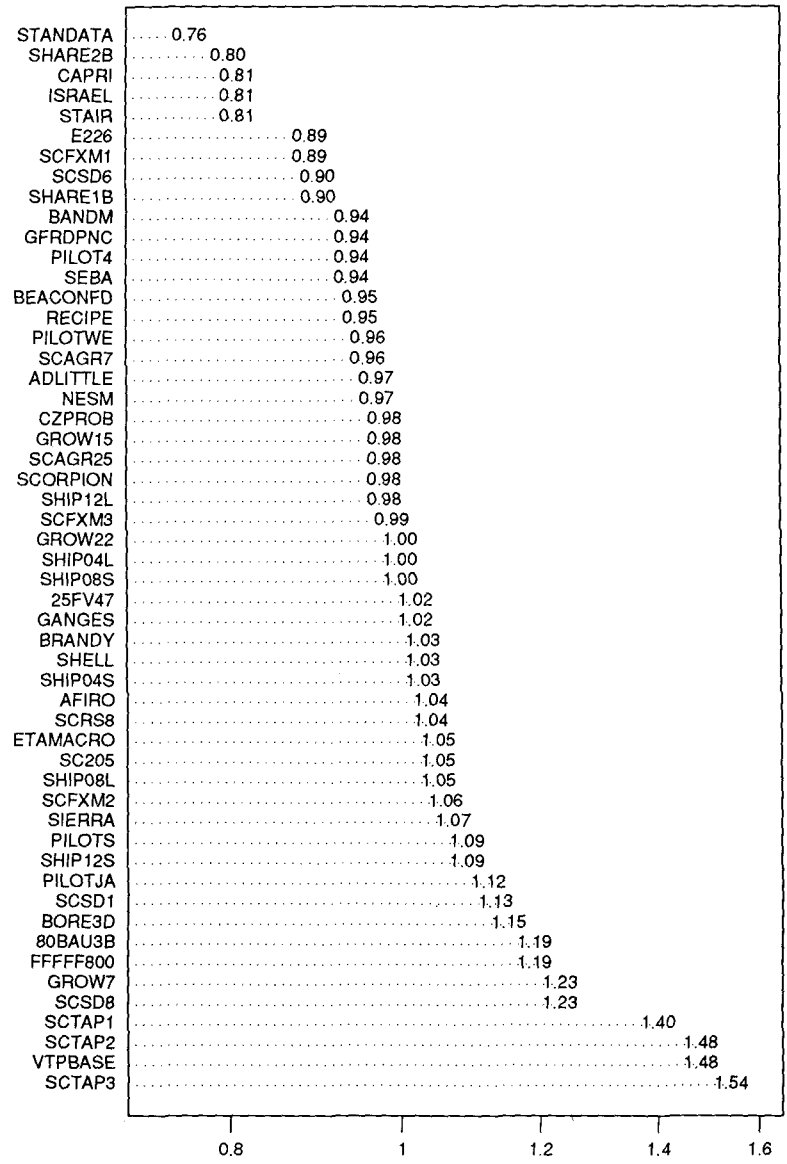

Fig. 8. Sorted comparison of solution times for SP2 and SP3.

10. Conclusions

The EXPAND procedure was developed in response to sporadic failures that occurred during Lustig's experiments with MINOS 5.1 on the 53 test problems used here [26]. No failures have occurred on these problems with the implementation described here.

Perhaps the main advance has been in the treatment of the infeasible blocking

variables generated by a Harris-type ratio test. Retaining numerical values when such variables become nonbasic means that $Ax = b$ can be satisfied to machine precision throughout, and allows full advantage to be taken of satisfying bounds loosely in the manner pioneered by Harris. An important benefit is that there is virtually no reversion to Phase 1 after refactorization—a common occurrence previously on ill-conditioned problems.

The precaution of expanding the feasibility tolerance at every iteration provides added theoretical protection against cycling (given the consequent similarity to Wolfe's anti-degeneracy procedure), as well as added practical assurance in the presence of rounding error.

Acknowledgments

The authors would like to express their appreciation to Irvin Lustig for his energetic experimentation during the summer of 1987. The present research, and the mechanisms for making numerous batch runs on multiple test problems, are a direct result. We are grateful to David Gay for making the test problems available through *netlib*. We thank the referees and especially Robert Fourer, whose comments were remarkably detailed and perceptive. Finally, we thank Jon Bentley for helpful suggestions concerning graphical presentation of results, and for the use of his "dot chart" package to produce Figures 4-8.

References

[1] M.L. Balinski and R.E. Gomory, "A mutual primal-dual simplex method," in: R.L. Graves and P. Wolfe, eds., *Recent Advances in Mathematical Programming* (McGraw-Hill, New York, 1963).

[2] E.M.L. Beale, "Advanced algorithmic features for general mathematical programming systems," in: J. Abadie, ed., *Integer and Nonlinear Programming* (North-Holland, Amsterdam, 1970) pp. 119-137.

[3] M. Benichou, J.M. Gauthier, G. Hentges and G. Ribière, "The efficient solution of large-scale linear programming problems—some algorithmic techniques and computational results," *Mathematical Programming* 13 (1977) 280-322.

[4] R.G. Bland, "New finite pivoting rules for the simplex method," *Mathematics of Operations Research* 2 (1977) 103-107.

[5] A. Brooke, D. Kendrick and A. Meeraus, *GAMS: A User's Guide* (The Scientific Press, Redwood City, CA, 1988).

[6] G.B. Dantzig, *Linear Programming and Extensions* (Princeton University Press, Princeton, NJ, 1963).

[7] G.B. Dantzig, "Making progress during a stall in the simplex algorithm," Report SOL 88-5, Department of Operations Research, Stanford University (Stanford, CA, 1988).

[8] G.B. Dantzig, A. Orden and P. Wolfe, "The generalized simplex method for minimizing a linear form under linear inequality constraints," *Pacific Journal of Mathematics* 5 (1955) 183-195.

[9] J.C. Falkner, "Bus crew scheduling and the set partitioning model," Ph.D. thesis, Department of Theoretical and Applied Mechanics, University of Auckland (Auckland, New Zealand, 1988).

[10] J.C. Falkner and D.M. Ryan, "Aspects of bus crew scheduling using a set partitioning model," Fourth International Workshop on Computer-Aided Scheduling of Public Transport (Hamburg, 1987).

[11] R. Fletcher, *Practical Methods of Optimization: Vol. 2: Constrained Optimization* (Wiley, Chichester and New York, 1981).

[12] R. Fletcher, "Degeneracy in the presence of round-off errors," Technical Report NA/89, Department of Mathematical Sciences, University of Dundee (Dundee, 1985).
[13] R. Fletcher, "Recent developments in linear and quadratic programming," in: A. Iserles and M.J.D. Powell, eds., *The State of the Art in Numerical Analysis* (Oxford University Press, Oxford and New York, 1987) pp. 213–243.
[14] R. Fletcher and M.P. Jackson, "Minimization of a quadratic function of many variables subject only to upper and lower bounds," *Journal of the Institute of Mathematics and its Applications* 14 (1974) 159–174.
[15] R. Fourer, "A simplex algorithm for piecewise-linear programming I: derivation and proof," *Mathematical Programming* 33 (1985) 204–233.
[16] R. Fourer and D.M. Gay, private communication (1989).
[17] D.M. Gay, "Electronic mail distribution of linear programming test problems," *Mathematical Programming Society COAL Newsletter* 13 (1985) 10–12.
[18] P.E. Gill, S.J. Hammarling, W. Murray, M.A. Saunders and M.H. Wright, "User's Guide for LSSOL (Version 1.0): a Fortran package for constrained linear least-squares and convex quadratic programming," Report SOL 86-1, Department of Operations Research, Stanford Univesity (Stanford, CA, 1986).
[19] P.E. Gill and W. Murray, "Numerically stable methods for quadratic programming," *Mathematical Programming* 14 (1978) 349–372.
[20] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright, "User's Guide for SOL/QPSOL (revised)," Report SOL 84-6, Department of Operations Research, Stanford University (Stanford, CA, 1984).
[21] P.E. Gill, W. Murray and M.H. Wright, *Practical Optimization* (Academic Press, London and New York, 1981).
[22] G.W. Graves, "A complete constructive algorithm for the general mixed linear programming problem," *Naval Research Logistics Quarterly* 12 (1965) 1–34.
[23] H.J. Greenberg, "Pivot selection tactics," in: H.J. Greenberg, ed., *Design and Implementation of Optimization Software* (Sijthoff and Noordhoff, Alphen aan den Rijn, 1978) pp. 143–174.
[24] P.M.J. Harris, "Pivot selection methods of the Devex LP code," *Mathematical Programming* 5 (1973) 1–28. [Reprinted in *Mathematical Programming Study* 4 (1975) 30–57.]
[25] E.S. Klotz, *Dynamic pricing criteria in linear programming*, Ph.D. thesis, Department of Operations Research, Stanford University (Stanford, CA, 1988).
[26] I.J. Lustig, "An analysis of an available set of linear programming test problems," Report SOL 87-11, Department of Operations Research, Stanford University (Stanford, CA, 1987). [See also *Computers and Operations Research* 16 (1989) 173–184.]
[27] B.A. Murtagh and M.A. Saunders, "MINOS 5.0 User's Guide," Report SOL 83-20, Department of Operations Research, Stanford University (Stanford, CA, 1983).
[28] B.A. Murtagh and M.A. Saunders, "MINOS 5.1 User's Guide," Report SOL 83-20R, Department of Operations Research, Stanford University (Stanford, CA, 1987).
[29] J.L. Nazareth, "Implementation aids for the optimization algorithms that solve sequences of linear programs," *ACM Transactions on Mathematical Software* 12 (1986) 307–323.
[30] J.L. Nazareth, *Computer Solution of Linear Programs* (Oxford University Press, New York and Oxford, 1987).
[31] W. Ogryczak, "On practical stopping rules for the simplex method," *Mathematical Programming Study* 31 (1987) 167–174.
[32] W. Orchard-Hays, *Advanced Linear-Programming Computing Techniques* (McGraw-Hill, New York, 1968).
[33] J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables* (Academic Press, London and New York, 1970).
[34] M.R. Osborne, *Finite Algorithms in Optimization and Data Analysis* (Wiley, New York, 1985).
[35] R.T. Rockafellar, *Network Flows and Monotropic Optimization* (Wiley, New York, 1984).
[36] D.M. Ryan and M.R. Osborne, "On the solution of highly degenerate linear programs," *Mathematical Programming* 41 (1988) 385–392.
[37] J.H. Wilkinson, *The Algebraic Eigenvalue Problem* (The Clarendon Press, Oxford, 1965).
[38] P. Wolfe, "The reduced-gradient method," unpublished manuscript, the RAND Corporation (1962).
[39] P. Wolfe, "A technique for resolving degeneracy in linear programming," *SIAM Journal of Applied Mathematics* 11 (1963) 205–211.
[40] P. Wolfe, "The composite simplex algorithm," *SIAM Review* 7 (1965) 42–54.