Open access • Journal Article • DOI:10.1137/050635432

Implementing Generating Set Search Methods for Linearly Constrained Minimization

Robert Michael Lewis, Anne Shepherd, Virginia Torczon

Institutions: College of William & Mary

Published on: 01 Oct 2007 - SIAM Journal on Scientific Computing (Society for Industrial and Applied Mathematics)

Topics: Feasible region, Constrained optimization, Set (abstract data type), Nonlinear programming and Computational geometry

IMPLEMENTING GENERATING SET SEARCH METHODS FOR LINEARLY CONSTRAINED MINIMIZATION*

ROBERT MICHAEL LEWIS†, ANNE SHEPHERD‡, AND VIRGINIA TORCZON§

Abstract. We discuss an implementation of a derivative-free generating set search method for linearly constrained minimization with no assumption of nondegeneracy placed on the constraints. The convergence guarantees for generating set search methods require that the set of search directions possesses certain geometrical properties that allow it to approximate the feasible region near the current iterate. In the hard case, the calculation of the search directions corresponds to finding the extreme rays of a cone with a degenerate vertex at the origin, a difficult problem. We discuss here how state-of-the-art computational geometry methods make it tractable to solve this problem in connection with generating set search. We also discuss a number of other practical issues of implementation, such as the careful treatment of equality constraints and the desirability of augmenting the set of search directions beyond the theoretically minimal set. We illustrate the behavior of the implementation on several problems from the CUTEr test suite. We have found it to be successful on problems with several hundred variables and linear constraints.

Key words. nonlinear programming, nonlinear optimization, constrained optimization, linear constraints, degeneracy, direct search, generating set search, generalized pattern search, derivative-free methods, double description algorithm

AMS subject classifications. 90C30, 90C56, 65K05

DOI. 10.1137/050635432

1. Introduction.

We consider ways to implement direct search methods for solving linearly constrained nonlinear optimization problems of the form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & \ell \le Ax \le u. \end{array} \tag{1.1}$$

The objective function is $f : \mathbb{R}^n \to \mathbb{R}$, with decision variables $x \in \mathbb{R}^n$. The constraint matrix is $A \in \mathbb{R}^{m \times n}$. If some constraints are unbounded below or above, the components of ℓ and u are allowed to take on the values $-\infty$ and $+\infty$, respectively. Our approach handles both equality and inequality constraints.

A key step in generating set search (GSS) algorithms [17] is the computation of the requisite search directions. Generating set search methods for linearly constrained problems achieve convergence without explicit recourse to the gradient or the directional derivative of the objective. They also do not attempt to estimate Lagrange multipliers. Instead, when the search is close to the boundary of the feasible

*Received by the editors July 7, 2005; accepted for publication (in revised form) May 31, 2007; published electronically DATE.

http://www.siam.org/journals/sisc/x-x/63543.html

†Department of Mathematics, College of William & Mary, P.O. Box 8795, Williamsburg, Virginia, 23187-8795; buckaroo@math.wm.edu. This research was supported by the National Aeronautics and Space Administration under Grant NCC-1-02029, by the Computer Science Research Institute at Sandia National Laboratories, and by the National Science Foundation under Grant DMS-0215444.

‡Department of Computer Science, College of William & Mary, P.O. Box 8795, Williamsburg, Virginia, 23187-8795; plshep@cs.wm.edu. This research was conducted under the appointment of a Department of Energy High-Performance Computer Science Graduate Fellowship administered by the Krell Institute and funded by the Computer Science Research Institute at Sandia National Laboratories.

§Department of Computer Science, College of William & Mary, P.O. Box 8795, Williamsburg, Virginia, 23187-8795; va@cs.wm.edu. This research was supported by the Computer Science Research Institute at Sandia National Laboratories.

region, the set of search directions must include directions that conform to the geometry of the nearby boundary. For linearly constrained minimization, this requires identifying a working set of nearly binding constraints at each iteration. The search directions then are computed as the generators of the polar of the cone generated by the outward-pointing normals to the constraints in the working set [24, 31, 21, 23, 18]. When the working set defines a cone with a nondegenerate vertex at the origin, the case is straightforward; computational approaches for handling this case have been known for some time [24, 31, 21].

The hard case arises if the working set defines a cone with a degenerate vertex at the origin. Analysis of algorithms that allow degenerate working sets is well-established [21, 23]. On the other hand, the resulting calculation of the search directions quickly becomes too difficult for a naive approach. As noted in [21], sophisticated and efficient computational geometry algorithms are required.

In this paper we show that by using state-of-the-art algorithms from computational geometry, the hard case can be dealt with in a computationally effective way and that the naive bounds on the combinatorial complexity involved in managing the hard case are unduly pessimistic. For example, in section 9 we illustrate a problem for which the estimate on the number of possible vectors to be considered is more than 2×10^{17} when, in fact, only 5 vectors are needed—and those vectors are correctly identified in less than one-fifth of a second on a standard desktop computer.

Our specific implementation makes use of the analysis of generating set search found in [18], which synthesizes the analytical results found in [21, 23]. A new feature of the algorithms presented in [18] is the way in which the working set of constraints is chosen, which we show here to be effective in practice.

We use Ω to denote the feasible region for problem (1.1):

$$\Omega = \{\, x \in \mathbb{R}^n \,:\, \ell \leq Ax \leq u \,\}.$$

The algorithm we present here, like all of the approaches referenced previously, is a *feasible iterates* method for solving (1.1): The initial iterate x_0 must be feasible, and all subsequent iterates x_k must satisfy $x_k \in \Omega$. We assume that f is continuously differentiable on Ω but that gradient information is not computationally available; i.e., no procedure exists for computing the gradient of the objective function, and it cannot be approximated accurately. We do not assume that the constraints are nondegenerate.

We discuss several topics regarding our implementation, though computing a sufficient set of search directions is the primary focus of this paper. Our goal is to dynamically identify a set of search directions sufficient to guarantee desirable convergence properties. The general idea is that the set must contain search directions that comprise all the generators for a particular cone of feasible directions, hence the name generating set search. An important question to be addressed in any implementation of this approach is what is meant by "close to the boundary"—i.e., which constraints belong in the current working set—since the current working set determines the requirements on the set of search directions needed for the iteration.

Once the current working set has been identified, the next question becomes how to obtain the desired set of search directions. If the current working set is not degenerate, then straightforward procedures are outlined in [24, 21]. In the presence of degeneracy, the situation becomes considerably more complex—a key issue we discuss here. Equalities in the working set were not treated explicitly in previous work, so here we elaborate on this case. We also discuss how augmenting the set of search directions can accelerate progress toward a solution.

Once a set of search directions has been identified, a step of appropriate length along each of the search directions must be found. The analysis makes this straightforward through the use of a scalar step-length control parameter Δ_k. In our implementation the choice of the working set is tied to Δ_k as proposed in [18].

Finally, there are the usual implementation details of optimization algorithms to consider. These include step acceptance criteria, stopping criteria, scaling of the variables, and data caching.

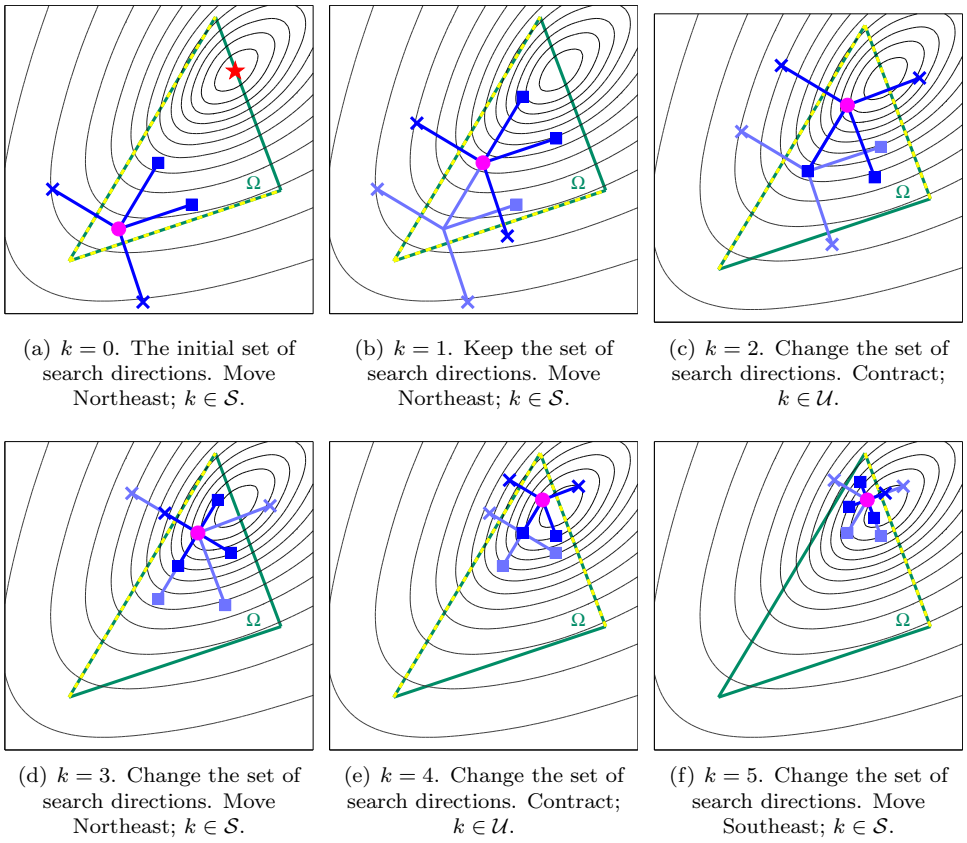
We begin in section 2 with an example and then give some preliminary definitions and notation in section 3. section 4 discusses how to identify the working set of constraints, while section 5 discusses how to generate the core set of search directions once the working set has been identified. section 6 discusses why it may be advantageous to augment the core set of search directions. section 7 discusses how we determine the lengths of the steps to be taken. In section 8 we state the algorithm and discuss other details such as initial feasible iterates (section 8.1), step-acceptance criteria (section 8.2), stopping conditions (section 8.3), scaling (section 8.4), and caching data (section 8.5). section 9 contains some illustrative numerical results. Concluding remarks are given in section 10.

2. An illustrative example. Figure 2.1 illustrates a few iterations of a greatly simplified linearly constrained GSS method applied to the two-dimensional modified Broyden tridiagonal function [4, 25], to which we have added three linear inequality constraints. Level curves of f are shown in the background. The three constraints form a triangle. In each figure, a dot denotes the current iterate x_k, which is the "best" point—the feasible point with the lowest value of f found so far.

We partition the indices of the iterations into two sets: $\mathcal{S}$ and $\mathcal{U}$. The set $\mathcal{S}$ corresponds to all *successful* iterations—those for which $x_k \neq x_{k+1}$ (i.e., at iteration k the search identified a feasible point with a lower value of the objective, and that point will become the next iterate). The set $\mathcal{U}$ corresponds to all *unsuccessful* iterations—those for which $x_{k+1} = x_k$ since the search was unable to identify a feasible point with a lower value of the objective.

Each of the six subfigures in Figure 2.1 represents one iteration of a linearly constrained GSS method. Taking a step of length Δ_k along each of the four search directions yields the four trial points. The squares represent the trial points under consideration (points at which the objective function is evaluated) at that particular iteration. The crosses represent the trial points that are not considered (points at which the objective function is not evaluated) at that particular iteration because they are infeasible. In subfigures (b)–(f), the trial points from the previous iteration are shown in the background for comparison. In subfigure (a), the solution to the problem is marked with a star.

In subfigure (a), the initial iterate x_0 is near two constraints, which are highlighted using dashed lines. (The mechanism by which we identify nearby constraints and its connection to the value of Δ_k are discussed in more detail in section 4.) Two of the initial search directions are chosen so that they are parallel to these two nearby constraints. The other two search directions are the normals to the nearby constraints, translated to x_0. In this instance, there are two feasible descent directions; i.e., moving along two of the four search directions would allow the search to realize descent on the value of f at x_0 while remaining feasible. Either of the two trial points produced by these two feasible descent directions is acceptable; here we choose the one that gives the larger decrease in the value of the objective f to become x_1. Since the step of length Δ_0 from x_0 to x_1 was sufficient to produce decrease, we set Δ_1 to Δ_0.



(a) $k = 0$. The initial set of search directions. Move Northeast; $k \in \mathcal{S}$.

(b) $k = 1$. Keep the set of search directions. Move Northeast; $k \in \mathcal{S}$.

(c) $k = 2$. Change the set of search directions. Contract; $k \in \mathcal{U}$.

(d) $k = 3$. Change the set of search directions. Move Northeast; $k \in \mathcal{S}$.

(e) $k = 4$. Change the set of search directions. Contract; $k \in \mathcal{U}$.

(f) $k = 5$. Change the set of search directions. Move Southeast; $k \in \mathcal{S}$.

FIG. 2.1. *A version of linearly constrained GSS applied to the modified Broyden tridiagonal function augmented with three linear inequality constraints.*

In subfigure (b), the same two constraints remain nearby, so we leave the set of search directions unchanged. Again, there are two feasible descent directions, each of which leads to a trial point that decreases the value of f at the current iterate. We accept the trial point that yields the larger decrease as x_2 and set Δ_2 to Δ_1.

In subfigure (c), the set of nearby constraints changes. We drop one of the constraints that had existed in our working set of nearby constraints in favor of a newly identified constraint that yields a different working set. The set of search directions that results is shown. We do have one descent direction, but now the difficulty is that currently the step along that direction takes the search outside the feasible region. Thus, the best iterate is unchanged, so we set x_3 to x_2. We then *contract* by setting Δ_3 to half the value of Δ_2, which has the effect of halving the length of the steps taken at the next iteration.

The consequences of the contraction are illustrated in subfigure (d). In addition to reducing the length of the step allowed, the contraction reduced the working set to one constraint. Given both the new set of search directions along with the reduction in the length of the steps allowed, we now have a feasible trial point that gives a decrease in the value of the objective, which we make x_4, and set Δ_4 to Δ_3.

In subfigure (e) the search moves to the new iterate. Once again the search is near

the same two constraints seen in subfigure (c), so we have the same working set and the same set of search directions, though now we search with a reduced step length (i.e., $\Delta_4 = \frac{1}{2}\Delta_2$). Again, the length of the steps along the two descent directions is too long. So we set x_5 to x_4 and Δ_5 to $\frac{1}{2}\Delta_4$.

The result of halving Δ is illustrated in subfigure (f). In addition to reducing the length of the steps allowed, we have changed the set of working constraints; once again, we have reduced the set to only one working constraint. A new feasible trial point that gives a decrease in the value of f is identified (to the Southeast), and at the next iteration the search will proceed from there.

This example illustrates the essential features the implementation must address: how to identify the nearby constraints, how to obtain a suitable set of feasible search directions, how to find a step of an appropriate length, and how to categorize the iteration as either successful ($k \in \mathcal{S}$) or unsuccessful ($k \in \mathcal{U}$).

3. Notation and definitions. Norms and inner products are assumed to be the Euclidean norm and inner product.

A *cone* K is a set that is closed under nonnegative scalar multiplication: K is a cone if $x \in K$ implies $ax \in K$ for all $a \geq 0$. A cone is *finitely generated* if there exists a (finite) set of vectors $v_1, \ldots, v_r$ such that

$$K = \{\ \lambda_1 v_1 + \cdots + \lambda_r v_r \ \mid\ \lambda_1, \ldots, \lambda_r \geq 0\ \}.$$

The vectors $v_1, \ldots, v_r$ are *generators* of K. Conversely, the cone generated by a set of vectors $v_1, \ldots, v_r$ is the set of all nonnegative combinations of these vectors.

The maximal linear subspace contained in a cone K is its *lineality space* [12]. The *polar* of a cone K, denoted by K°, is the set

$$K^\circ = \left\{\ v \ \mid\ w^T v \leq 0 \text{ for all } w \in K\ \right\}.$$

The polar K° is a convex cone, and if K is finitely generated, then so is K°.

Let K be a convex cone. Given a vector v, let v_K and v_{K° denote the projections of v onto K and its polar K°, respectively. The polar decomposition [26, 32] says that any vector v can be written as the sum of its projections onto K and K° and that the projections are orthogonal: $v = v_K + v_{K^\circ}$, and $v_K^T v_{K^\circ} = 0$. This means that $\mathbb{R}^n$ is positively spanned by a set of generators of K together with a set of generators of K°, a generalization of the direct sum decomposition given by a subspace and its orthogonal complement.

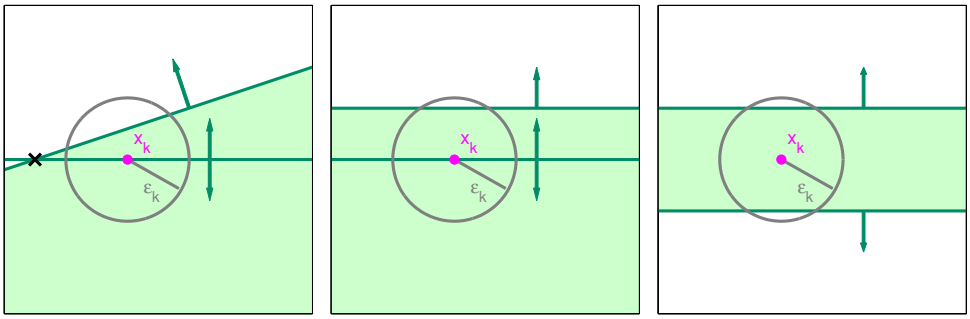
We borrow the following definition from [18]. For any finite set of vectors $\mathcal{G}$,

$$\kappa(\mathcal{G}) = \inf_{\substack{v \in \mathbb{R}^n \\ v_K \neq 0}} \max_{d \in \mathcal{G}} \frac{v^T d}{\| v_K \| \, \| d \|}, \quad \text{where } K \text{ is the cone generated by } \mathcal{G}. \tag{3.1}$$

The value $\kappa(\mathcal{G})$ is a property of the set $\mathcal{G}$, not of the cone K. The measure $\kappa(\mathcal{G})$ plays a critical role when choosing the set of search directions, as discussed further in section 5. Given (3.1), we then have the following.

PROPOSITION 3.1 (Proposition 10.3 of [21]). *If $\mathcal{G} \neq \{\mathbf{0}\}$, then $\kappa(\mathcal{G}) > 0$.*

If K is a vector space, then a set of generators for K is called a *positive spanning set* [7]. A positive spanning set is like a linear spanning set but with the additional requirement that all of the coefficients be nonnegative.



(a) The inequality constraint is not included in the working set since its intersection with the equality constraint lies outside the ε_k-ball.

(b) The inequality constraint is redundant given the equality constraint.

(c) Both the upper and lower bounds are in the working set.

FIG. 4.1. *Special cases for linear equality constraints.*

4. Constructing the working set of constraints. We next discuss how we determine the working set of constraints that is used to compute the minimal set of necessary search directions. Let a_i^T denote the rows of A, indexed by i. We partition the constraints into the equalities and inequalities:

$$\mathcal{E} = \{\ i\ \mid\ \ell_i = u_i\ \} \quad \text{and} \quad \mathcal{I} = \{\ i\ \mid\ \ell_i < u_i\ \}.$$

We denote by $\mathcal{C}_{\ell,i}$ and $\mathcal{C}_{u,i}$ the sets where the bounds on constraint i are binding:

$$\mathcal{C}_{\ell,i} = \left\{\ y\ \mid\ a_i^T y = \ell_i\ \right\} \quad \text{and} \quad \mathcal{C}_{u,i} = \left\{\ y\ \mid\ a_i^T y = u_i\ \right\}.$$

These sets are faces of the feasible polyhedron. Of interest to us are the outward-pointing normals to the faces within a prescribed distance of x.

The presence of equality constraints slightly complicates the identification of inequality constraints near a given iterate. Figure 4.1(a) illustrates the desirability of measuring the distances to nearly binding inequality constraints only along directions that are feasible with respect to equality constraints. The horizontal line in Figure 4.1(a) represents an equality constraint, while the line at an oblique angle represents an inequality constraint. The shaded region represents the set of points feasible with respect to the inequality constraint. The inequality constraint is in close proximity to the equality constraint near x but only at points that are infeasible with respect to the equality constraint (a cross indicates where both constraints are active).

We compute the distance to nearby binding inequality constraints along directions that are feasible with respect to equality constraints as follows. Let $\mathcal{N}$ denote the nullspace of the equality constraints (i.e., the matrix whose rows are a_i^T for $i \in \mathcal{E}$), and let Z be an $n \times r$ orthogonal matrix whose columns are a basis for $\mathcal{N}$. Given $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$, let $S = \left\{\ y\ \mid\ a^T y = b\ \right\}$. We then define

$$\text{dist}_{\mathcal{N}}(x, S) = \begin{cases} |\, a^T x - b\,| / \| \, Z^T a \, \| & \text{if } Z^T a \neq 0, \\ 0 & \text{if } Z^T a = 0 \text{ and } a^T x = b, \\ \infty & \text{if } Z^T a = 0 \text{ and } a^T x \neq b. \end{cases}$$

This is the distance from x to S inside the nullspace $\mathcal{N}$. If S is parallel to $\mathcal{N}$, as is the case for the inequality constraint in Figure 4.1(b), we set this distance to be ∞,

unless $x \in S$, in which case the distance is 0, as is true for the equality constraint in Figure 4.1(b).

Using this specialized notion of distance, we define the sets of inequalities at their bounds that are within distance ε of x:

$$\mathcal{I}_\ell(x,\varepsilon) = \{\ i \in \mathcal{I}\ \mid\ \mathrm{dist}_{\mathcal{N}}(x, \mathcal{C}_{\ell,i}) \le \varepsilon\ \},$$
$$\mathcal{I}_u(x,\varepsilon) = \{\ i \in \mathcal{I}\ \mid\ \mathrm{dist}_{\mathcal{N}}(x, \mathcal{C}_{u,i}) \le \varepsilon\ \}.$$

Let $\mathcal{I}_E(x,\varepsilon) = \mathcal{I}_\ell(x,\varepsilon) \cap \mathcal{I}_u(x,\varepsilon)$.

We now define the working set of constraints. Given a feasible x and an $\varepsilon > 0$, the corresponding working set $\mathcal{W}(x,\varepsilon)$ is made up of two pieces, the first consisting of vectors we treat as defining equalities and the second consisting of vectors we treat as defining inequalities. The first piece is

$$\mathcal{W}_E(x,\varepsilon) = \{\ a_i\ \mid\ i \in \mathcal{E}\ \} \cup \{\ a_i\ \mid\ i \in \mathcal{I}_E(x,\varepsilon)\ \},$$

which corresponds to the equality constraints together with every inequality for which the faces for both its lower and upper bounds are within distance ε of x, as in Figure 4.1(c). The second piece is

$$\mathcal{W}_I(x,\varepsilon) = \{\ -a_i\ \mid\ i \in \mathcal{I}_\ell(x,\varepsilon) \setminus \mathcal{I}_E(x,\varepsilon)\ \} \cup \{\ a_i\ \mid\ i \in \mathcal{I}_u(x,\varepsilon) \setminus \mathcal{I}_E(x,\varepsilon)\ \},$$

which is the set of outward-pointing normals to the inequalities for which the faces of exactly one of their lower or upper bounds is within distance ε of x. Note that, for the three examples illustrated in Figure 4.1, $\mathcal{W}_I(x,\varepsilon) = \emptyset$. The distinction between $\mathcal{W}_E$, which consists of constraints that are treated as equalities, and $\mathcal{W}_I$, which consists of constraints that are treated as inequalities, figures in the computation of the requisite search directions. We return to this point in section 5.

Given $x \in \Omega$, the *ε-normal cone*, denoted $N(x,\varepsilon)$, is the cone generated by the linear span of the vectors in $\mathcal{W}_E(x,\varepsilon)$ together with the nonnegative span of the vectors in $\mathcal{W}_I(x,\varepsilon)$:

$$N(x,\varepsilon) = \left\{ \sum_{a_i \in \mathcal{W}_E(x,\varepsilon)} u_i a_i + \sum_{a_i \in \mathcal{W}_I(x,\varepsilon)} \xi_i a_i\ \mid\ \xi_i \ge 0 \right\}. \tag{4.1}$$

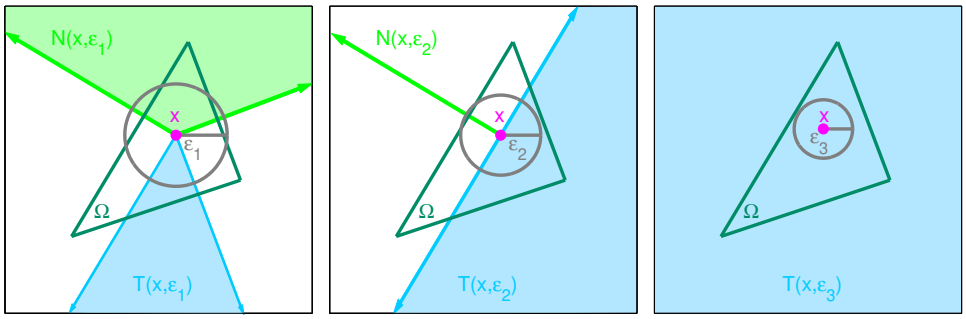
If both of the latter sets are empty, then $N(x,\varepsilon) = \{\mathbf{0}\}$. The *$\varepsilon$-tangent cone*, denoted by $T(x,\varepsilon)$, is the polar of $N(x,\varepsilon)$:

$$T(x,\varepsilon) = \left\{\ v\ \mid\ w^T v \le 0 \text{ for all } w \in N(x,\varepsilon)\ \right\}.$$

If $N(x,\varepsilon) = \{\mathbf{0}\}$, then $T(x,\varepsilon) = \mathbb{R}^n$. In addition to the examples illustrated in Figure 4.1, three examples involving only inequality constraints with three different values of ε are illustrated in Figure 4.2.

The set $x + T(x,\varepsilon)$ approximates the feasible region near x, where "near" is in terms of ε as measured using $\mathrm{dist}_{\mathcal{N}}(x,S)$. The cone $T(x,\varepsilon)$ is important because if $T(x,\varepsilon) \ne \{\mathbf{0}\}$, then one can proceed from x along all directions in $T(x,\varepsilon)$ for a distance of at least ε and still remain inside the feasible region [21, 18]. The examples in Figures 4.1–4.2 illustrate this point.

In the algorithm, we allow different values of ε_k at every iteration k. There is a lot of flexibility in determining the choice of ε_k, as discussed in [18]. We have chosen to yoke ε_k to Δ_k, for the reasons given in [18].



(a) Cones for ε_1. (b) Cones for ε_2. (c) Cones for ε_3.

FIG. 4.2. *The cones $N(x,\varepsilon)$ and $T(x,\varepsilon)$ for three values of ε. Note that for this example, as ε varies from ε_1 to 0, there are only the three distinct pairs of cones ($N(x,\varepsilon_3) = \{\mathbf{0}\}$).*

5. Generating the core set of search directions. Once the nearby constraints for a given iteration have been identified, we select the search directions. To ensure that the convergence results in [18] hold, we make the search conform to the local geometry of Ω in the sense that, at each iteration k, there is at least one direction in $\mathcal{D}_k$, the set of search directions at iteration k, along which the search can take an acceptably long step and still remain feasible.

We decompose $\mathcal{D}_k$ into $\mathcal{G}_k$ and $\mathcal{H}_k$, where $\mathcal{G}_k$ is made up of the directions necessary to ensure desirable convergence properties and $\mathcal{H}_k$ contains any additional directions. We sometimes refer to $\mathcal{G}_k$ as the *core set* of search directions. Several options for choosing $\mathcal{G}_k$ have been proposed:

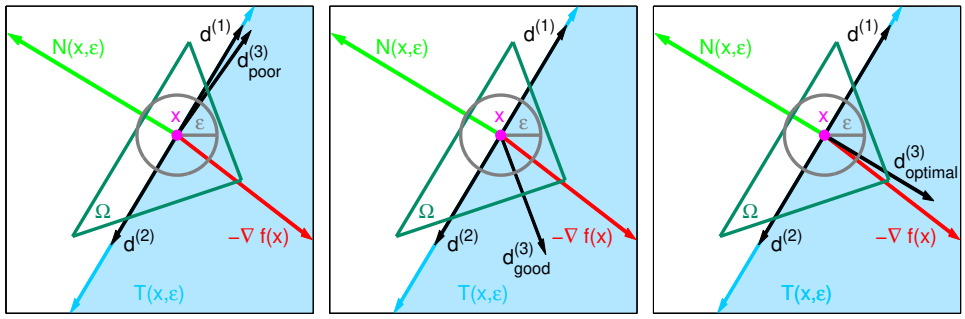
1. $\mathcal{G}_k$ contains generators for all of the cones $T(x_k,\varepsilon)$ for all $\varepsilon \in [0,\varepsilon_*]$, where $\varepsilon_* > 0$ is independent of k [24, 21].
2. $\mathcal{G}_k$ is exactly the set of generators for the cone $T(x_k,\varepsilon_k)$, where $\varepsilon_k \to 0$ is updated according to whether acceptable steps are found, and $\mathcal{H}_k = \emptyset$ [23].
3. $\mathcal{G}_k$ contains generators for the cone $T(x_k,\varepsilon_k)$, where $\varepsilon_k = \Delta_k\beta_{\max}$, and $\beta_{\max}$ is described in [18].

Here we use the third option, with its explicit link between ε_k and Δ_k, and choose $\beta_{\max} = 1$. As discussed further in section 6, the third option allows us to use a nonempty set $\mathcal{H}_k$ of additional directions chosen so as to accelerate the overall progress of the search. For any of the three options, the following is required of $\mathcal{G}_k$ [21, 18]:

> CONDITION 1. There exists a constant $\kappa_{\min} > 0$, independent of k, such that, for every k for which $T(x_k,\varepsilon_k) \neq \{\mathbf{0}\}$, the set $\mathcal{G}_k$ generates $T(x_k,\varepsilon_k)$ and satisfies $\kappa(\mathcal{G}_k) \geq \kappa_{\min}$.

Observe that the number of distinct ε-normal cones (and consequently the number of distinct ε-tangent cones) is finite.

The following simple technique, outlined in [18], ensures Condition 1 is satisfied. Let $k_2 > k_1$. If $\mathcal{W}_E(x_{k_2},\varepsilon_{k_2}) = \mathcal{W}_E(x_{k_1},\varepsilon_{k_1})$ and $\mathcal{W}_I(x_{k_2},\varepsilon_{k_2}) = \mathcal{W}_I(x_{k_1},\varepsilon_{k_1})$, then use the same generators for $T(x_{k_2},\varepsilon_{k_2})$ as were used for $T(x_{k_1},\varepsilon_{k_1})$. There are at most 2^m distinct sets $\mathcal{G}$ if the same set of generators is always used to generate a particular ε-tangent cone. Recall that m is the number of linear constraints. Since by Proposition 3.1 each $\mathcal{G}_k \neq \{\mathbf{0}\}$ has a strictly positive value for $\kappa(\mathcal{G}_k)$, and since this



(a) Poor choice (b) Reasonable choice (c) "Optimal" choice (w.r.t. $\kappa(\mathcal{G}_k)$)

FIG. 5.1. *Condition 1 is needed to avoid a poor choice when choosing* $\mathcal{G}_k$.

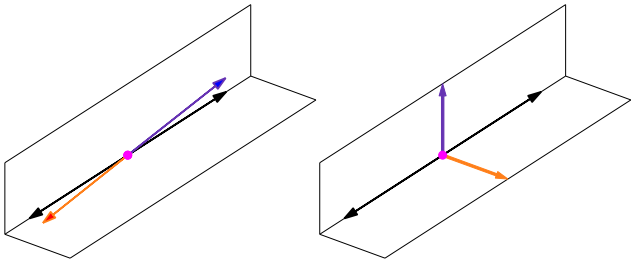


FIG. 5.2. *On the right, a set of generators for the cone (which includes a lineality space along the fold) that satisfies Condition 1 with a relatively large value of* $\kappa_{\min}$. *On the left, a different set of generators for which* $\kappa_{\min}$ *is much closer to zero.*

technique ensures there are only finitely many $\mathcal{G}_k$'s, we can set $\kappa_{\min} = \min\{\kappa(\mathcal{G}_k) : T(x_k, \varepsilon_k) \neq \{\mathbf{0}\}\}$. Thus, Condition 1 is satisfied.

The need for Condition 1 arises when there is a lineality space present in $T(x_k, \varepsilon_k)$. In this case the directions of the generators $\mathcal{G}_k$ are not uniquely determined. Figure 5.1 depicts the situation where $T(x_k, \varepsilon_k)$ contains a halfspace. Condition 1 enforces a lower bound on $\kappa(\mathcal{G}_k)$ to avoid the situation illustrated in Figure 5.1(a), in which the search directions can be almost orthogonal to the direction of steepest descent. Figure 5.1(c), which uses a minimal number of vectors but ensures as large a value for $\kappa(\mathcal{G}_k)$ as possible, represents an optimal choice in the absence of explicit knowledge of the gradient.

In $\mathbb{R}^2$ the presence of a lineality space in $T(x, \varepsilon)$ means that $T(x, \varepsilon)$ is either a halfspace, as illustrated in Figures 4.2(b) and 5.1, or $\mathbb{R}^2$, as illustrated in Figure 4.2(c). In more than two dimensions there are other geometrical possibilities that cause the directions of the generators of $T(x, \varepsilon)$ not to be uniquely determined, as illustrated in Figure 5.2.

As discussed in [18], we need to enforce lower and upper bounds on the lengths of the search directions. Here we accomplish this by simply normalizing all of the search directions.

The necessary conditions on the set of search directions all involve the calculation of generators for the polars of cones determined by the current working set. We treat the determination of the search directions in increasing order of difficulty. Figure 5.3

shows the various cases to consider.

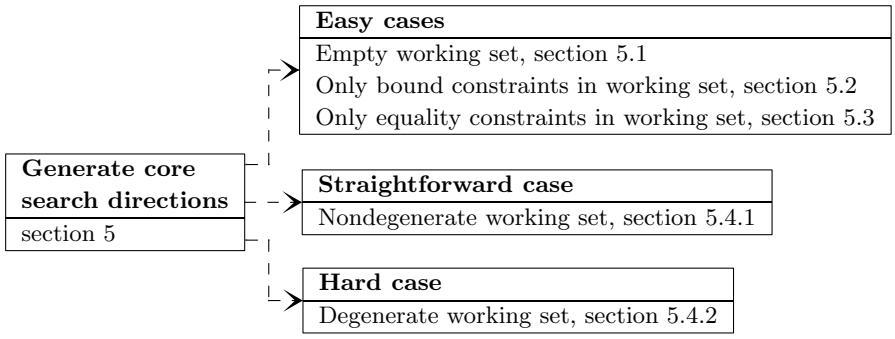


FIG. 5.3. *The components involved in the generation of the search directions.*

5.1. When the working set is empty. If the working set is empty, then $N(x_k, \varepsilon_k) = \{\mathbf{0}\}$ and $T(x_k, \varepsilon_k) = \mathbb{R}^n$, and any positive spanning set of $\mathbb{R}^n$ suffices as $\mathcal{G}_k$ [30, 19]. We use the unit coordinate vectors $\pm e_i$, $i = 1, \ldots, n$, to form $\mathcal{G}_k$ both because they are apt for many applications and because the orthonormality of the vectors ensures a reasonable value for $\kappa_{\min}$ in Condition 1. In our testing we have found that the coordinate directions are a consistently effective set of search directions for scientific and engineering problems. The coordinate directions have physical meaning for such applications; thus using them to form the core set of search directions $\mathcal{G}_k$ both guarantees a positive basis and seems to lead to better overall performance.

5.2. When the working set contains only bound constraints. A similar situation holds if only bound constraints are present. This case is also simple since we know a priori that the generators for the ε-normal and ε-tangent cones can be drawn from the set of coordinate directions [20]. Thus we include in $\mathcal{G}_k$ the unit coordinate vectors $\pm e_i$, $i = 1, \ldots, n$. As noted in [21, section 8.3], if not all of the variables are bounded, then one can make a choice of $\mathcal{G}_k$ that is more parsimonious in the number of directions, but we did not choose to do so in our implementation.

5.3. When the working set consists only of equality constraints. If the working set consists only of equality constraints, then the generators of $T(x_k, \varepsilon_k)$ correspond to a positive spanning set for the nullspace of the vectors in $\mathcal{W}_E(x_k, \varepsilon_k)$. In this situation by default we compute an orthonormal basis Z for this nullspace and take as our generators the "coordinatelike" set consisting of the elements of Z and their negatives.

5.4. When the working set contains general linear constraints. Once the nearby constraints have been identified using x_k and ε_k, their outward-pointing normals are extracted from the rows of the constraint matrix A to form the working set of constraints. So, for instance, in the case shown in Figure 5.4 the outward-pointing normals a_1 and a_2 constitute the working set $\mathcal{W}_I(x_k, \varepsilon_k) = \{a_1, a_2\}$ (there are no equality constraints in this example). There are then two cases to consider: when the working set is known to be nondegenerate and when the working set may be degenerate.

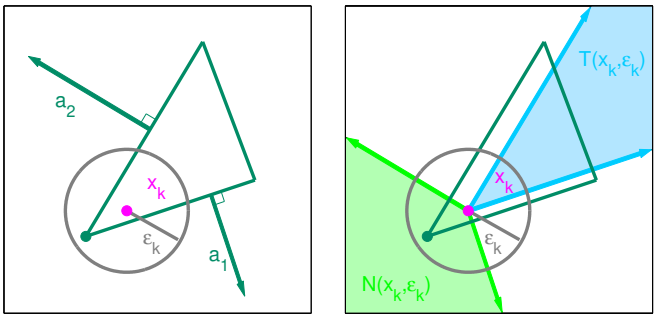


FIG. 5.4. *The outward-pointing normals a_1 and a_2 of the nearby constraints are translated to x_k to form the ε-normal cone $N(x_k, \varepsilon_k)$ and thus define its polar the ε-tangent cone $T(x_k, \varepsilon_k)$. The normalized generators for $N(x_k, \varepsilon_k)$ and $T(x_k, \varepsilon_k)$ form the set of search directions.*

5.4.1. When the working set is known to be nondegenerate. Figure 5.4 depicts the computationally straightforward case.

We begin with two general propositions concerning the polar of a finitely generated cone K. Proposition 5.1 is an elaboration of the construction given in [21, section 8.2] and shows how one can construct generators for K° if the generators of K are linearly independent. In particular, this means that K cannot contain a linear subspace. In those situations where K does contain a linear subspace, for instance, as illustrated in Figures 5.1–5.2, Proposition 5.2 enables us to reduce the situation to one where we can apply Proposition 5.1.

PROPOSITION 5.1. *Suppose K is generated by the positive span of the columns of the matrix $Q = [q_1 \dots q_r]$. Let N be a matrix whose columns are a positive spanning set for the nullspace of Q^T. Finally, suppose Q^T has a right inverse R.*

Then K° is the positive span of the columns of $-R$ together with the linear span of the columns of N:

$$K^\circ = \{\ w\ \mid\ w = -Ru + N\xi,\quad u \geq 0\ \}.$$

Proof. Let $C = \{\ w\ \mid\ w = -Ru + N\xi,\quad u \geq 0\ \}$. We first show that $C \subset K^\circ$. Suppose $d \in C$. To show that $d \in K^\circ$, we must show that $q_i^T d \leq 0$ for all $i = 1, \dots, r$, i.e., $Q^T d \leq 0$. Since $Q^T R = I$ and $Q^T N = 0$,

$$Q^T d = Q^T(-Ru + N\xi) = -u \leq 0.$$

Therefore $d \in K^\circ$, so $C \subset K^\circ$.

We next show that $C^\circ \subset K$, whence $K^\circ \subset C$, and so $K^\circ = C$. If $d \in C^\circ$, then $d^T(-Ru + N\xi) \leq 0$ for $u \geq 0$ and for all ξ. In particular, $d^T N\xi = 0$ for all ξ, so d is orthogonal to the positive span of the columns of N, which is the nullspace of Q^T. The latter means that d lies in the range of Q, so $d = Qs$ for some s. At the same time, we know that $-d^T Ru \geq 0$ for all $u \geq 0$, whence $-s^T Q^T Ru = -s^T u \leq 0$ for all $u \geq 0$, so $s \geq 0$. Thus $d = Qs$ for $s \geq 0$, meaning $d \in K$ and $C^\circ \subset K$. ∎

Proposition 5.2 says that if the cone K contains a linear subspace L, then computing K° is a matter of looking at the polar of $K \cap L^\perp$.

PROPOSITION 5.2. *Suppose that the cone K can be written as the linear span of the vectors $v_1, \dots, v_q$ together with the nonnegative span of the vectors $p_1, \dots, p_r$:*

$$K = \{\ v\ \mid\ v = \alpha_1 v_1 + \dots + \alpha_q v_q + \lambda_1 p_1 + \dots + \lambda_r p_r,\ \lambda_i \geq 0 \text{ for } i = 1, \dots, r\ \}.$$

Step 1. Let L be the linear subspace spanned by the vectors in $\mathcal{W}_E(x_k, \varepsilon_k)$. Compute a basis Z for $L^\perp$.

Step 2. Check whether the set of vectors $\{ Z^T p \mid p \in \mathcal{W}_I(x_k, \varepsilon_k) \}$ is linearly independent. If so, proceed to Step 3. If not, proceed to the more complicated construction described in section 5.4.2.

Step 3. Let Q be the matrix whose columns are the vectors $Z^T p$ for $p \in \mathcal{W}_I(x_k, \varepsilon_k)$. Compute a right inverse R for Q^T and a matrix N whose columns are a positive spanning set for the nullspace of Q^T.

Step 4. $\mathcal{G}_k$ is then the columns of $-ZR$ together the columns of ZN.

FIG. 5.5. *The process for attempting the straightforward construction of* $\mathcal{G}_k$.

Let L be the linear subspace spanned by $v_1, \ldots, v_q$. Then

$$K^\circ = \{ w \in L^\perp \mid w^T p_i \leq 0 \text{ for all } i = 1, \ldots, r \}.$$

Proof. Let $P = \{ w \in L^\perp \mid w^T p_i \leq 0 \text{ for all } i = 1, \ldots, r \}$. Clearly, $P \subset K^\circ$. Conversely, if $w \in K^\circ$, then we have $w^T p_i \leq 0$ for all $i = 1, \ldots, r$. In addition, since L is contained in K we have $v_i \in K$ and $-v_i \in K$ for all $i = 1, \ldots, q$. Thus we have $w^T v_i \leq 0$ and $-w^T v_i \leq 0$, so $w^T v_i = 0$ and $w \in L^\perp$, whence $K^\circ \subset P$. ∎

We use Propositions 5.1 and 5.2 to compute generators for $T(x_k, \varepsilon_k) = N(x_k, \varepsilon_k)^\circ$ as follows. Recall the definition of $N(x_k, \varepsilon_k)$ in (4.1). Let L denote the linear span of the vectors in $\mathcal{W}_E(x_k, \varepsilon_k)$. Let Z denote a matrix whose columns span $L^\perp$. Applying Proposition 5.2 with $K = N(x_k, \varepsilon_k)$ then says

$$\begin{aligned} T(x_k, \varepsilon_k) &= \{ w \in L^\perp \mid w^T p \leq 0 \text{ for all } p \in \mathcal{W}_I(x_k, \varepsilon_k) \} \\ &= \{ Zc \mid c^T Z^T p \leq 0 \text{ for all } p \in \mathcal{W}_I(x_k, \varepsilon_k) \}. \end{aligned}$$

The latter condition says that the set of c of interest to us is actually the polar of the cone $\tilde{K}$ generated by the vectors $Z^T p$ for all $p \in \mathcal{W}_I(x_k, \varepsilon_k)$. If this latter set of vectors is linearly independent, we can apply Proposition 5.1 to obtain generators for $\tilde{K}^\circ$ and then map them back to the original space under the action of Z. If they are not linearly independent, we must use the more sophisticated approach discussed in section 5.4.2. The procedure is summarized in Figure 5.5.

In our implementation we compute an orthonormal basis Z in Step 1. In Step 3 we use the pseudoinverse of Q^T as the right inverse. This choice is attractive because the columns of the pseudoinverse are orthogonal to the lineality space of $T(x_k, \varepsilon_k)$ (as in the examples on the right in Figures 5.1–5.2), which helps improve the conditioning of the resulting set of search directions (Condition 1).

Also in Step 3 our implementation computes the positive spanning set N by computing an orthonormal basis B for the nullspace of Q^T and setting $N = [B \ -B]$. This corresponds to a coordinatelike search in the nullspace of Q^T.

5.4.2. When the working set may be degenerate. If the construction described in Figure 5.5 cannot be applied, because linear dependence of the set of vectors $\{ Z^T p \mid p \in \mathcal{W}_I(x_k, \varepsilon_k) \}$ is detected, the computation of the search directions

is more complicated. In this case $N(x_k, \varepsilon_k)$ is a cone with a degenerate vertex at the origin, so we refer to this situation as the degenerate case. (The easiest example to envision is the apex defined by the four triangular faces of a pyramid.)

The problem of finding a set of generators for $T(x_k, \varepsilon_k)$ is a special case of the problem of finding the vertices and extreme rays of a polyhedron (the V-representation of the polyhedron) given the description of the polyhedron as an intersection of halfspaces (the H-representation). This is a well-studied problem in computational geometry (see [2] for a survey) for which practical algorithms have been developed.

The two main classes of algorithms for solving this type of problem are pivoting algorithms and insertion algorithms. Pivoting algorithms move from vertex to vertex along the edges of the polyhedron by generating feasible bases, as in the simplex method. The reverse search algorithm [3, 1] is an example of such a method.

Insertion algorithms compute the V-description by working with the intersection of an increasing number of halfspaces. Typically, for a polyhedral cone these algorithms start with a maximal linearly independent set of normals defining the halfspaces and construct an initial set of extreme rays as in Figure 5.5. Additional halfspaces are then introduced, and the extreme rays are recomputed as positive combinations of existing extreme rays; this process continues until the entire set of halfspaces has been introduced. The double description method [27] is an example of an insertion algorithm. A discussion of the double description method, along with a description of the techniques used in a practical implementation, can be found in [11].

As observed in [2], pivoting algorithms can be inefficient if they encounter a highly degenerate vertex. For this reason we chose the double description method. We use Fukuda's `cddlib` package [10], which is based on an efficient implementation of the double description algorithm that directly addresses known—and serious—pitfalls of a naive implementation of the double description method and that has been shown to be effective in dealing with highly degenerate vertices [11]. We conducted our own comparison of Avis's `lrs` reverse search code [1] and Fukuda's `cdd` double description code [10] on problems we knew to be highly degenerate. Our experience confirmed the observation made in [9]: `cdd` tends to be efficient for highly degenerate inputs, while `lrs` tends to be efficient for nondegenerate or slightly degenerate problems.

Another observation made in [9] that bears repeating is that the number of inequalities defining the polyhedron does not by itself indicate the degree of difficulty of a vertex/ray enumeration problem. It is noted in [9] that `cdd` can handle a highly degenerate problem with 711 inequalities in 19 dimensions quite easily, while a problem with 729 inequalities in 8 dimensions is extremely difficult to solve. Our own experience confirms this, as discussed further in section 9.

During our testing with `cdd`, one useful lesson we learned is that it is more efficient to retain equalities in the problem rather than to eliminate the equalities by reparameterizing the problem in terms of a basis for the nullspace of the equality constraints. We initially thought that using the equalities to eliminate variables and reduce the dimension of the problem would be an advantage. But this did not prove to be the case. In several instances, elimination of equalities led to a significant deterioration in the performance of `cdd`. This is due to the fact that the arrays of constraint coefficients went from being sparse—and frequently integral—to being dense real arrays, which greatly hindered the progress of the double description method.

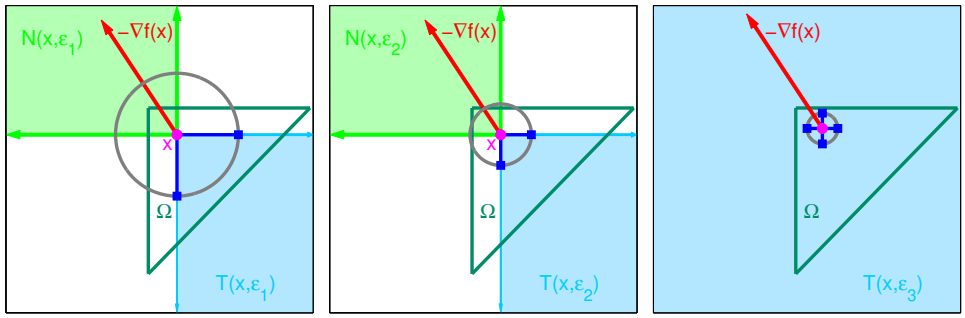
We close by noting that only a subset of the generators for $T(x_k, \varepsilon_k)$ may suffice at some iterations. For example, if an iteration is successful, the iteration can be terminated as soon as at least one of the search directions yields a trial point that

satisfies the acceptance criterion. Thus, the generators could be computed on an "as needed" basis (though, for an unsuccessful iteration, ultimately a complete set of generators for $T(x_k, \varepsilon_k)$ will be required). Such an approach would be akin to that used by the pivoting algorithms. But, as already noted, the pivoting algorithms can be inefficient if they encounter a highly degenerate vertex. Since `cdd` has proven efficient on all but one of the problems we have tested, we currently compute all of the generators for $T(x_k, \varepsilon_k)$ at each iteration for which the working set changes.

6. Augmenting the core set of search directions. While generators of $T(x_k, \varepsilon_k)$ are theoretically sufficient for the convergence analysis, in practice there can be appreciable benefits to including additional search directions. The situation illustrated in Figure 6.1 shows the desirability of additional directions, since the generators for $T(x_k, \varepsilon_k)$ necessarily point into the relative interior of the feasible region, while it may be preferable to move toward the boundary.

At the very least, we would like the total set of search directions to be a positive basis for $\mathbb{R}^n$ or, if equality constraints are present, a positive basis for the nullspace of the equality constraints. If $\mathcal{W}_E(x_k, \varepsilon_k)$ is empty, one simple way to accomplish this is to include in $\mathcal{H}_k$ the set $\mathcal{W}_I(x_k, \varepsilon_k)$, the outward-pointing normals to the working set of constraints. Since these vectors generate $N(x_k, \varepsilon_k)$, and since $T(x_k, \varepsilon_k)$ is polar to $N(x_k, \varepsilon_k)$, by the polar decomposition, $\mathbb{R}^n = N(x_k, \varepsilon_k) \oplus T(x_k, \varepsilon_k)$. Thus, including the vectors that generate $N(x_k, \varepsilon_k)$ in $\mathcal{H}_k$ means that $\mathcal{D}_k$ constitutes a positive spanning set for $\mathbb{R}^n$, as illustrated in Figure 6.2. See section 9 for a computational illustration of the advantage of including these directions.

The preceding discussion needs to be modified if equality constraints are present. Typically, in this case the outward-pointing normals a_i to constraints in the working set are infeasible directions—they do not lie in the nullspace of the equality constraints. Thus, a step along a direction a_i will lead to violation of the equality constraints. In this situation we replace the outward-pointing normals by their projection into the nullspace of the equality constraints.



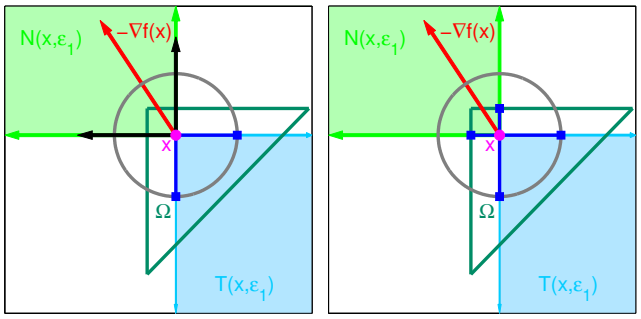
(a) First iteration is unsuccessful and so Δ_1 is halved for the next iteration.

(b) The second iteration is also unsuccessful and so Δ_2 is halved again.

(c) Now, locally, the problem looks unconstrained and a successful step toward the boundary is possible.

FIG. 6.1. *Minimizing the number of function evaluations at a single iteration might require more total function evaluations since in this instance it will take at least five function evaluations to find decrease.*

7. Determining the lengths of the steps. The *step-length control parameter* Δ_k determines the length of a step along each direction for a given set of search



(a) Both generators of $N(x, \varepsilon_1)$ are descent directions.

(b) Limiting the steps along the generators of $N(x, \varepsilon_1)$ ensures feasible steps.

FIG. 6.2. *Why including the generators for the ε-normal cone $N(x, \varepsilon_1)$ might be a good idea since decrease may be found after as few as one function evaluation.*

directions. In the presence of constraints, we must consider the possibility that some of the trial points $x_k + \Delta_k d_k^{(i)}$, $d_k^{(i)} \in \mathcal{D}_k$, may be infeasible. With this in mind, we choose the maximum $\tilde{\Delta}_k^{(i)} \in [0, \Delta_k]$ such that $x_k + \tilde{\Delta}_k^{(i)} d_k^{(i)} \in \Omega$, thereby ensuring that a full step will be taken, if feasible.

It is possible to choose $\tilde{\Delta}_k^{(i)} = \Delta_k$ if the result will be feasible and $\tilde{\Delta}_k^{(i)} = 0$ otherwise, thereby rejecting infeasible points. This simple strategy was illustrated in the initial example given in Figure 2.1. We have found it much more efficient, however, to allow steps that stop at the boundary, as suggested in [22, 23]. The latter strategy is illustrated in Figure 6.2(b).

For the search directions in $\mathcal{H}_k$ we actually choose $\tilde{\Delta}_k^{(i)}$ to be the maximum value in $[\sigma_{\text{tol}} \Delta_k, \Delta_k]$ satisfying $x_k + \tilde{\Delta}_k^{(i)} d_k^{(i)} \in \Omega$, where $\sigma_{\text{tol}} \geq 0$. The factor σ_{tol} prevents excessively short steps to the boundary, relative to the current value of Δ_k. Our experience has been that such steps typically yield little improvement and have the overall effect of retarding decrease in Δ_k.

8. Linearly constrained GSS. The variant of GSS that we have implemented for problems with linear constraints is outlined in Figure 8.1. See [18] for discussions of some of the many other possible variations on the specification of GSS algorithms for linearly constrained optimization. The remainder of this section is devoted to further explanation of the algorithm given in Figure 8.1.

8.1. Finding an initial feasible iterate. GSS methods for linearly constrained problems are feasible iterates methods—all iterates x_k, $k = 0, 1, 2, \ldots$, satisfy $x_k \in \Omega$. If an infeasible initial iterate is given, then we compute its Euclidean projection onto the feasible polyhedron; this is a quadratic program. For some of the test problems discussed in section 9 the starting value provided is infeasible, and for these we used this projection strategy. Alternatively, one could use phase one of the simplex method to obtain a starting point at a vertex of the feasible region.

8.2. Accepting steps. We opted to impose the sufficient decrease condition

$$f(x_k + \tilde{\Delta}_k d_k) < f(x_k) - \rho(\Delta_k),$$

ALGORITHM 8.1 (LINEARLY CONSTRAINED GSS USING A SUFFICIENT DECREASE GLOBALIZATION STRATEGY)

INITIALIZATION.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be given. Let Ω be the feasible region.

Let $x_0 \in \Omega$ be the initial guess.

Let $\Delta_{\text{tol}} > 0$ be the step-length convergence tolerance.

Let $\Delta_0 > \Delta_{\text{tol}}$ be the initial value of the step-length control parameter.

Let $\sigma_{\text{tol}} \geq 0$ control the lower bound on the length of successful steps.

Let $\varepsilon_{\max} > \Delta_{\text{tol}}$ be the maximum distance used to identify nearby constraints ($\varepsilon_{\max} = +\infty$ is permissible).

Let $\rho(\Delta) = \alpha\Delta^2$ be the forcing function, with $\alpha > 0$.

ALGORITHM. For each iteration $k = 0, 1, 2, \ldots$

Step 1. Let $\varepsilon_k = \min\{\varepsilon_{\max}, \Delta_k\}$. Choose a set of normalized search directions $\mathcal{D}_k = \mathcal{G}_k \cup \mathcal{H}_k$ satisfying Condition 1.

Step 2. If there exists $d_k^{(i)} \in \mathcal{D}_k$, together with the maximum $\tilde{\Delta}_k^{(i)} \in [\sigma_{\text{tol}}\Delta_k, \Delta_k]$ such that $x_k + \tilde{\Delta}_k d_k \in \Omega$, which satisfy

$$f\left(x_k + \tilde{\Delta}_k^{(i)} d_k^{(i)}\right) < f(x_k) - \rho(\Delta_k),$$

then:

- Set $x_{k+1} = x_k + \tilde{\Delta}_k^{(i)} d_k^{(i)}$ (a successful step).
- Set $\Delta_{k+1} = \Delta_k$.

Step 3. Otherwise, for every $d \in \mathcal{G}_k$,

$$f(x_k + \Delta_k d) \geq f(x_k) - \rho(\Delta_k).$$

In this case:

- Set $x_{k+1} = x_k$ (an unsuccessful step).
- Set $\Delta_{k+1} = \frac{1}{2}\Delta_k$.

If $\Delta_{k+1} < \Delta_{\text{tol}}$, then terminate.

FIG. 8.1. *Linearly constrained GSS using a sufficient decrease globalization strategy.*

with $\rho(\Delta) = \alpha\Delta^2$. For any given problem the value of α should be chosen to reflect the relative magnitude of f and Δ. Our primary reason for using a sufficient decrease condition was that, as discussed in section 7 (and illustrated in Figure 6.2(b)), it allows us to choose $\tilde{\Delta}_k^{(i)}$ to be the maximum value in $[\sigma_{\text{tol}}\Delta_k, \Delta_k]$ satisfying $x_k + \tilde{\Delta}_k^{(i)} d_k^{(i)} \in \Omega$. Furthermore, as shown in [18, section 6], this particular choice of $\rho(\Delta)$

ensures the useful estimate (8.1), which we discuss in the next section.

For the results reported here we used the classical update for Δ_k. If $k \in \mathcal{S}$, then $\Delta_{k+1} = \phi_k \Delta_k$, with $\phi_k = 1$ for all k; otherwise, $\Delta_{k+1} = \theta_k \Delta_k$, with $\theta_k = \frac{1}{2}$ for all k.

8.3. Stopping criteria. We stop the algorithm once the step-length parameter Δ_k falls below a specified tolerance Δ_{tol}. This stopping criterion is motivated by the relationship between Δ_k and $\chi(x_k)$, a particular measure of progress toward a Karush–Kuhn–Tucker (KKT) point of (1.1). For $x \in \Omega$, let

$$\chi(x) \equiv \max_{\substack{x+w \in \Omega \\ \| w \| \leq 1}} -\nabla f(x)^T w.$$

Of interest to us here are the following properties [5, 6]: (1) $\chi(x)$ is continuous on Ω, (2) $\chi(x) \geq 0$, and (3) $\chi(x) = 0$ if and only if x is a KKT point for (1.1). We therefore wish to see $\chi(x_k)$ tend to zero.

In [18, Theorem 6.4] it is shown that once Δ_k is small enough, so that $\varepsilon_k = \Delta_k$, then

$$\chi(x_k) = O(\Delta_k) \text{ for } k \in \mathcal{U}. \tag{8.1}$$

Theorem 6.4 is most simply stated under the assumptions that the set $\mathcal{F} = \{ x \in \Omega \mid f(x) \leq f(x_0) \}$ is bounded and that the gradient of f is Lipschitz continuous with constant M on $\mathcal{F}$. Relaxations of these assumptions, along with a full statement and proof of the theorem, are found in [18].

The big-O relationship between $\chi(x_k)$ and Δ_k means that as Δ_k is reduced, which happens only at unsuccessful iterations, the upper bound on the value of the measure of stationarity is also reduced. Relationship (8.1) is analogous to the unconstrained minimization result (see [8, section 3] or [17, section 3.6]):

$$\| \nabla f(x_k) \| = O(\Delta_k) \text{ for } k \in \mathcal{U}. \tag{8.2}$$

Results (8.1) and (8.2) support using the magnitude of Δ_k as a test for termination.

Obviously, one can also stop after a specified number of objective evaluations, an option we exercise in our testing in section 9.

8.4. Scaling. The relative scaling of variables can have a profound effect on the efficiency of optimization algorithms. We allow as an option the common technique of shifting and rescaling the variables so that they have a similar range and size. We work in a computational space whose variables w are related to the original variables x via $x = Dw + c$, where D is a diagonal matrix with positive diagonal entries and c is a constant vector, both provided by the user. We solve the transformed problem

$$\begin{aligned} &\text{minimize} && f(Dw + c) \\ &\text{subject to} && \ell - Ac \;\leq\; ADw \;\leq\; u - Ac. \end{aligned}$$

The scaling scheme we use in the numerical tests presented in section 9 is based on that described in [13, section 7.5.1]. The latter scaling is based on the range of the variables and transforms the variables to lie between -1 and 1.

One advantage of scaling is that it makes it easier to define a reasonable default value for Δ_0. For the results reported in section 9, whenever scaling is applied to a problem, the default value for Δ_0 is 2 since the longest possible feasible step along a unit coordinate direction is 2 (the variables in the rescaled problem lie between -1 and 1).

8.5. Caching data. The potentially regular pattern to the steps the search takes means that the search may revisit points. This is particularly true for unconstrained and bound constrained problems. To avoid reevaluating known values of f, we store values of x at which we evaluate the objective and the corresponding objective values. Before evaluating f at a trial point w_t in the rescaled space, we examine the data cache to see whether we have already evaluated f at a point within some relative distance of w_t. We do not evaluate f at w_t if we have already evaluated f at a point $\bar{w}$ for which $\| w_t - \bar{w} \| < r_{\text{tol}} \| w_t \|$. By default, $r_{\text{tol}} = 10^{-8}$. For a more sophisticated caching strategy, see [16].

Caching yields only minor improvements in efficiency if general linear inequality constraints are present. If there are a considerable number of general linear constraints (i.e., constraints other than bounds) in the working set, then there is only a small chance of reevaluating f near a previous iterate. Still, if each evaluation of the objective function is expensive, then it is worth avoiding reevaluations.

9. Some numerical illustrations. We present some illustrations of the behavior of our implementation of the algorithm given in Figure 8.1 using five test problems from the CUTEr test suite [14, 15], listed in Table 9.1. All five problems have nonlinear objectives. We chose these specific problems for illustration since the algorithm must deal repeatedly with highly degenerate instances of the search direction calculation discussed in section 5.4.2. As the results in section 9.2 show, the solution of such problems is usually quite tractable given a good implementation of the double description algorithm.

9.1. Starting values and tolerances for the results reported here. We started with the value of x_0 specified by CUTEr, though we often had to project into the feasible region, as discussed in section 8.1.

We terminated the program either when the value of Δ_k fell below $\Delta_{\text{tol}} = \Delta_0/2^{20}$ or when the number of evaluations of the objective reached that which would have been needed for 20 centered difference evaluations of the gradient if one had used the equality constraints to eliminate variables. See Table 9.2 for the maximum number of evaluations allowed for each problem.

When the problem was scaled (as it was for all but LOADBAL), we used $\Delta_0 = 2$, as discussed in section 8.4. We used the default value $\sigma_{\text{tol}} = 10^{-3}$. We set $\varepsilon_{\max} = 2^5\Delta_0$ so that it played no role in the results reported here. We set $\alpha = 10^{-4}$ in the forcing function $\rho(\Delta) = \alpha\Delta^2$.

The core set of search directions $\mathcal{G}_k$ consisted of generators for the ε-tangent cone $T(x_k, \varepsilon_k)$. With one exception (for the purposes of illustrating the potential advantage

		# of linear			# of bounds	
Problem	**n**	**Network equalities**	**Equalities**	**Inequalities**	**Lower**	**Upper**
AVION2	49	0	15	0	49	49
DALLASM	197	151	0	0	197	197
DALLASS	46	31	0	0	46	46
LOADBAL	31	11	0	20	31	11
SPANHYD	97	33	16	0	97	97

TABLE 9.1
Problems used from the CUTEr test problem set

Value of objective/Value of Δ	
○	$f(x_k)$, $k \in \mathcal{S}$ (*successful*)
●	$f(x_k)$, $k \in \mathcal{U}$ (*unsuccessful*)
△	magnitude of Δ_k
—	optimal objective value (computed using MINOS [28])

Number of directions	
○	cardinality of the working set; no change in $\mathcal{D}_k$
●	cardinality of the working set; `cddlib` called
○	cardinality of the working set; otherwise
□	cardinality of $\mathcal{G}_k$ (*set of core directions*)
✳	cardinality of $\mathcal{H}_k$ (*set of extra directions*)
—	n (number of decision variables)

FIG. 9.1. *Legends for the remaining figures.*

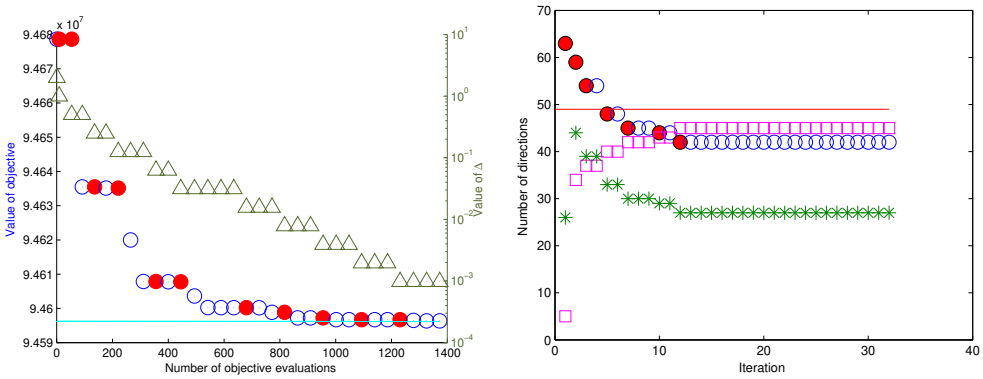


FIG. 9.2. *Results for the* AVION2 *test problem. Legends are given in Figure* 9.1.

of allowing a nonempty set $\mathcal{H}_k$) $\mathcal{H}_k$ consisted of generators for the ε-normal cone $N(x_k, \varepsilon_k)$ (projected into the nullspace of any equality constraints in the working set, as described in section 6). Recall that, by definition, $\mathcal{D}_k = \mathcal{G}_k \cup \mathcal{H}_k$.

The tests were run on a 1 processor, 3 GHz Pentium 4 workstation with 2 GB memory running Linux and using Matlab R2006b.

9.2. The results. Below we include a summary, along with illustrations, of the the iteration history for each problem. See Figure 9.1 for a legend.

The first problem is AVION2, an aeronautical design problem. In this case the equality constraints have rank 15, leaving 34 degrees of freedom in the problem. The plot on the left in Figure 9.2 gives the iteration history for the value of the objective and the value of Δ, both plotted against the number of evaluations of the objective. Solid circles indicate where unsuccessful iterations were encountered. The plot on the right in Figure 9.2 shows the size of the working set at each iteration, the number of vectors in $\mathcal{G}_k$, and the number of vectors in $\mathcal{H}_k$. Solid circles in this plot indicate iterations where the simple polar construction in section 5.4.1 was not applicable and consequently `cddlib` was called, as described in section 5.4.2. Open circles indicate iterations where the working set was unchanged from the previous iteration so no new calculation of the search directions was needed. The set $\mathcal{H}_k$ has fewer vectors than the working set because we do not include outward-pointing normals for equality constraints in the working set. The solid line indicates the number of decision variables in the problem; when the size of the working set is above this line, the algorithm must deal with the degeneracy discussed in section 5.4.2. Observe that degeneracy is also encountered when the size of the working set is smaller than the number of decision variables; this occurs because of redundant constraints.

The next test problem is SPANHYD, a model of a hydroelectric system. This problem includes 33 network equality constraints of rank 32 and 16 fixed variables,

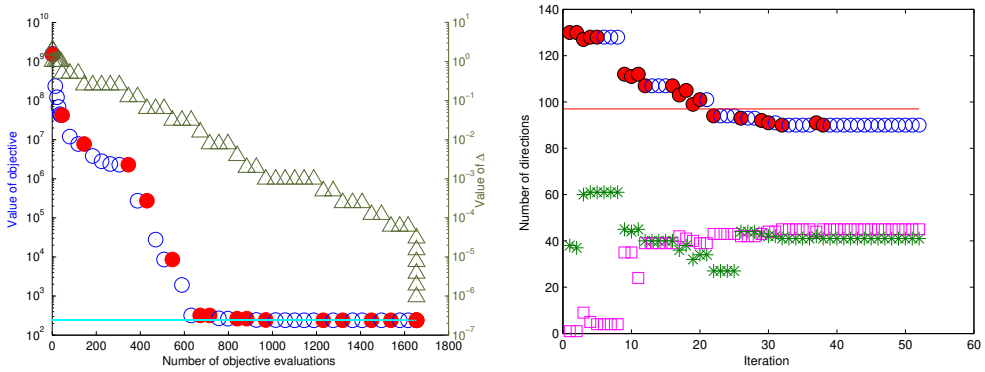


FIG. 9.3. *Results for the* SPANHYD *test problem. Legends are given in Figure* 9.1.

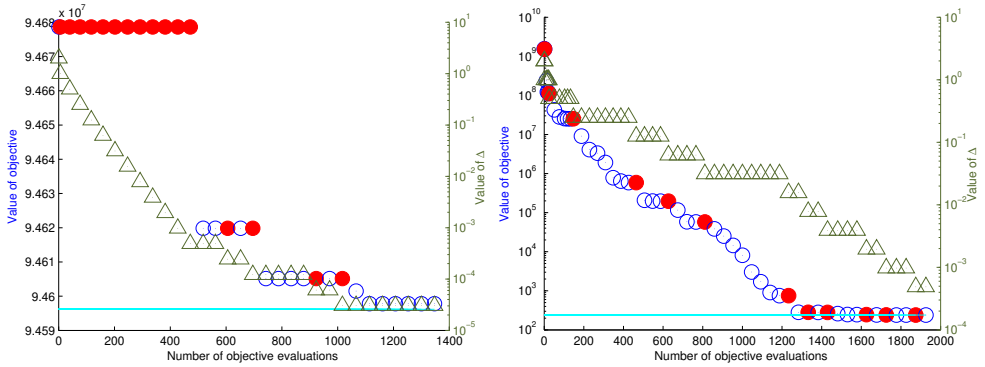


FIG. 9.4. *Progress versus cost history for the* AVION2 *(left) and* SPANHYD *(right) test problems if only the theoretically minimal set of search directions* $\mathcal{D}_k = \mathcal{G}_k$ *is used at each iteration* k*. Legend is given in Figure* 9.1.

leaving 49 degrees of freedom in the problem. We modify the scaling scheme slightly by not scaling variables whose original range is narrow; specifically, we scale only the variables that vary over a range of more than 0.1. The plot on the left in Figure 9.3 shows that the algorithm makes good progress toward the optimal objective value. For this problem the test terminated because $\Delta_k \leq \Delta_{\text{tol}}$.

More interesting is the plot on the right, which shows that the degenerate case arises multiple times. Moreover, the plot on the right in Figure 9.3 reveals how badly one can be fooled by the naive bound on the potential number of search directions. For instance, at iteration 4 there are 128 constraints in the working set. Of these, 67 are equality constraints, with rank 66. Eliminating these degrees of freedom from the 97 variables still leaves brute force enumeration of

$$\begin{pmatrix} 128-67 \\ 97-66 \end{pmatrix} = \frac{61!}{31!\ 30!} \approx 2.33 \times 10^{17}$$

possible generators for $T(x_k, \varepsilon_k)$. There are, in fact, only 5 generators, which `cddlib` finds in less than 0.17 seconds.

Figure 9.4 illustrates the point made in section 6 about the desirability of using more than the theoretically minimal set of search directions. In this test we used only the generators of $T(x_k, \varepsilon_k)$ (the core set of search directions), setting $\mathcal{H}_k =$

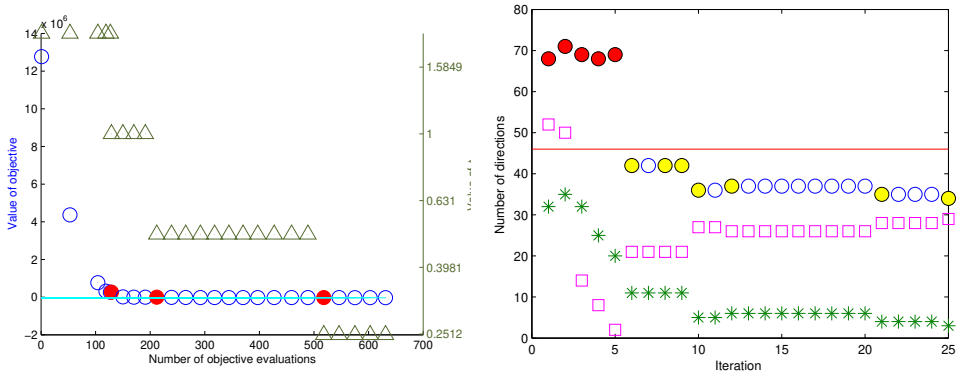


FIG. 9.5. *Results for the* DALLASS *test problem. Legends are given in Figure* 9.1.

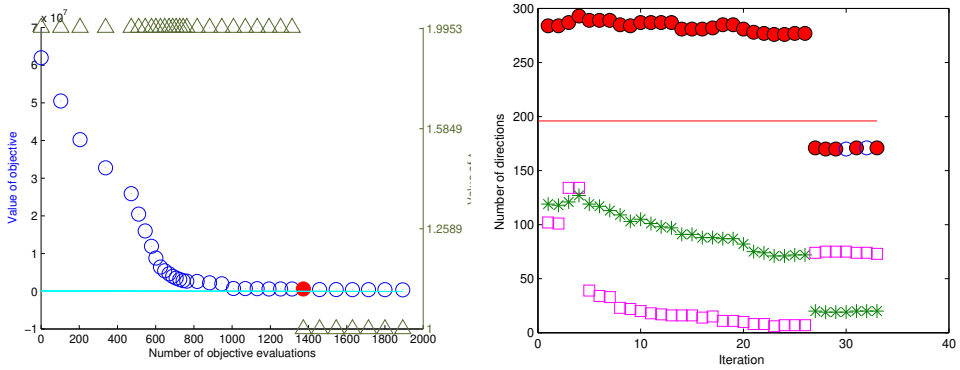


FIG. 9.6. *Results for the* DALLASM *test problem. Legends are given in Figure* 9.1.

$\emptyset$. Comparisons with Figures 9.2 and 9.3 show that using the smaller set of search directions can appreciably retard the progress of the search.

Figure 9.5 gives results for the DALLASS test problem, a small model of the Dallas water system. The plot on the left shows that the problem is solved quickly. In the plot on the right, lighter solid circles indicate iterations where the simpler construction of the search directions given in section 5.4.1 was used. The degenerate case occurs only in the first five iterations.

In Figure 9.6 we give the results for the DALLASM test problem, a medium size model of the Dallas water system. While the algorithm makes progress towards the solution, reducing the value of the objective by a factor of 160, it does not solve the problem within the budget of objective evaluations allowed. The plot on the right shows that in the first 26 iterations there is considerable degeneracy in the working set. In the most extreme case, at iteration 4 there are 293 constraints in the working set, of which 151 are equalities with rank 150. Eliminating these degrees of freedom from the 197 variables leaves

$$\begin{pmatrix} 293 - 151 \\ 197 - 150 \end{pmatrix} = \frac{142!}{47!\ 95!} \approx 1.01 \times 10^{38}$$

possible generators for $T(x_k, \varepsilon_k)$. There are actually only 134 generators, which `cddlib` finds in about 4.3 seconds.

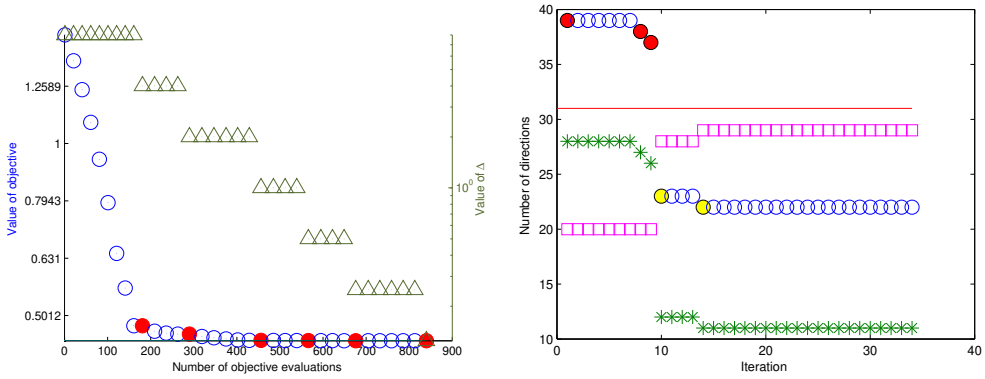


FIG. 9.7. *Results for the* LOADBAL *test problem. Legends are given in Figure* 9.1.

CUTEr problem	Degrees of freedom	Stopping conditions		Total time (seconds)	
		Max evals.	Reason	Elapsed	In `cddlib`
AVION2	34	1380	Max evals.	2.47	0.15
DALLASM	47	1900	Max evals.	92.91	75.86
DALLASS	16	660	Max evals.	1.32	0.09
LOADBAL	21	860	Max evals.	1.56	0.02
SPANHYD	49	1980	$\Delta < \Delta_{\text{tol}}$	6.68	2.43

TABLE 9.2
Summary of test results

Figure 9.7 gives results for the LOADBAL problem, a load–balancing problem from computer science. Absent upper bounds on all of the variables, and with no prior knowledge of reasonable ranges for the variables, we do not scale. We choose $\Delta_0 = 8$ based on the magnitude of the starting point. There is rapid progress toward the solution.

We summarize the test results in Table 9.2.
In evaluating the timings, one should bear in mind that the cost of an objective evaluation was insignificant since the objectives are explicitly defined functions. In more realistic settings the cost of an objective evaluation would likely be much higher, so the relative cost of treating degenerate working sets should be appreciably less.

We tested our implementation on other problems from the CUTEr test set (e.g., HIMMELBI) and for most of them saw results comparable to the ones reported above. Two exceptions were QPCBOEI1 and QPCBOEI2. In both cases we encountered highly degenerate cases of the working set that exercised `cddlib`. For instance, in QPCBOEI1, a 384 variable problem, we encountered working sets with over 700 constraints for which `cddlib` took about 40 seconds to compute the core search directions. In QPCBOEI2, a 143 variable problem, we encountered a working set consisting of 30 equalities and 189 inequalities for which `cddlib` did not find a set of core search directions in an acceptable amount of time; we terminated the job after allowing `cddlib` to work for 12 hours.

10. Concluding remarks. Our numerical results illustrate the value of the condition defining the working set for generating set search introduced in [18]. This condition reduces the number of core search directions required in [21] while allowing

the flexibility of introducing extra search directions. As the discussion in section 6 would suggest, we have found that augmenting the set of search directions beyond the theoretically minimal set typically leads to greater overall efficiency of the algorithm. We also have shown that state-of-the-art computational geometry methods make the calculation of the search directions needed by generating set search tractable even in the degenerate case. In addition, the presentation here addresses the various details that arise when handling linear equality constraints. Finally, the introduction of linear constraints has allowed us to solve larger dimensional problems than is usually possible for the unconstrained case.

Our implementation gives us a basis for implementing a generating set search version of the augmented Lagrangian algorithm in [5]. In this approach, linear constraints are treated explicitly rather than being included in the augmented Lagrangian. The ability to treat degeneracy explicitly, as we do here, is important because the combination of general linear constraints and simple bounds frequently leads to degeneracy, and treatment of degeneracy by an augmented Lagrangian approach can be slow.

Future work includes using information collected during the progress of the algorithm to estimate the set of constraints binding at a solution as well as developing strategies for moving more quickly to solutions on the boundary.

Acknowledgments. Komei Fukuda has made his `cdd` package publicly available and graciously answered our questions about `cddlib`. Conversations with Rakesh Kumar about his experience implementing and testing the algorithm in [21] for the MathWorks Genetic Algorithm and Direct Search Toolbox [29] provided many useful insights when we tackled degenerate constraints. Josh Griffin read an early version of the paper and reported back numerous bugs and inconsistencies that we had overlooked. Tammy Kolda, Associate Editor, and two anonymous referees carefully read the paper and responded with numerous suggestions that appreciably improved the presentation. Stephen Nash also sent along several helpful observations concerning the first version of the paper. We extend our sincere thanks to all of these people.

REFERENCES

[1] D. Avis, *lrs: A revised implementation of the reverse search vertex enumeration algorithm*, in Polytopes - Combinatorics and Computation, G. Kalai and G. Ziegler, eds., Birkhauser-Verlag, 2000, pp. 177–198.

[2] D. Avis, D. Bremner, and R. Seidel, *How good are convex hull algorithms?*, Computational Geometry: Theory and Applications, 7 (1997), pp. 265–301.

[3] D. M. Avis and K. Fukuda, *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*, Discrete and Computational Geometry, 8 (1992), pp. 295–313.

[4] C. G. Broyden, *A class of methods for solving nonlinear simultaneous equations*, Mathematics of Computation, 19 (1965), pp. 577–593.

[5] A. R. Conn, N. Gould, A. Sartenaer, and P. L. Toint, *Convergence properties of an augmented Lagrangian algorithm for optimization with a combination of general equality and linear constraints*, SIAM Journal on Optimization, 6 (1996), pp. 674–703.

[6] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, vol. 1 of MPS/SIAM Series on Optimization, SIAM, Philadelphia, 2000.

[7] C. Davis, *Theory of positive linear dependence*, American Journal of Mathematics, 76 (1954), pp. 733–746.

[8] E. D. Dolan, R. M. Lewis, and V. J. Torczon, *On the local convergence properties of pattern search*, SIAM Journal on Optimization, 14 (2003), pp. 567–583.

[9] K. Fukuda, *cdd/cdd+ Reference Manual*, Institute for Operations Research, ETH–Zentrum, Zurich, Switzerland, 2005. Available at `ftp://ftp.ifor.math.ethz.ch/pub/fukuda/cdd/cddman/cddman.html`.

[10] ———, *cddlib*, 2005. `http://www.ifor.math.ethz.ch/~fukuda/cdd_home/`.

[11] K. Fukuda and A. Prodon, *Double description method revisited*, in Combinatorics and Computer Science, M. Deza, R. Euler, and I. Manoussakis, eds., vol. 1120 of Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 91–111.

[12] M. Gerstenhaber, *Theory of convex polyhedral cones*, in Activity Analysis of Production and Allocation, T. C. Koopmans, ed., John Wiley & Sons, New York, 1951, pp. 298–316. Cowles Commission Monograph No. 13.

[13] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London, 1981.

[14] N. I. M. Gould, D. Orban, and P. L. Toint, *CUTEr: a Constrained and Unconstrained Testing Environment, revisited*, 2001. http://cuter.rl.ac.uk/cuter-www/doc.html.

[15] N. I. M. Gould, D. Orban, and P. L. Toint, *CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited*, ACM Transactions on Mathematical Software, 29 (2003), pp. 373–394.

[16] P. D. Hough, T. G. Kolda, and H. A. Patrick, *Usage manual for APPSPACK 2.0*, Tech. Rep. SAND2000–8843, Sandia National Laboratories, Livermore, CA 94550, August 2000.

[17] T. G. Kolda, R. M. Lewis, and V. Torczon, *Optimization by direct search: New perspectives on some classical and modern methods*, SIAM Review, 45 (2003), pp. 385–482.

[18] ———, *Stationarity results for generating set search for linearly constrained optimization*, SIAM Journal on Optimization, 17 (2006), pp. 943–968.

[19] R. M. Lewis and V. Torczon, *Rank ordering and positive bases in pattern search algorithms*, Tech. Rep. 96–71, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia, 1996.

[20] ———, *Pattern search algorithms for bound constrained minimization*, SIAM Journal on Optimization, 9 (1999), pp. 1082–1099.

[21] ———, *Pattern search methods for linearly constrained minimization*, SIAM Journal on Optimization, 10 (2000), pp. 917–941.

[22] S. Lucidi and M. Sciandrone, *A derivative-free algorithm for bound constrained optimization*, Computational Optimization and Applications, 21 (2002), pp. 119–142.

[23] S. Lucidi, M. Sciandrone, and P. Tseng, *Objective-derivative-free methods for constrained optimization*, Mathematical Programming, 92 (2002), pp. 37–59.

[24] J. H. May, *Linearly Constrained Nonlinear Programming: A Solution Method That Does Not Require Analytic Derivatives*, PhD thesis, Yale University, New Haven, Connecticut, December 1974.

[25] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, Association for Computing Machinery (ACM) Transactions on Mathematical Software, 7 (1981), pp. 17–41.

[26] J.-J. Moreau, *Décomposition orthgonale d'un espace hilbertien selon deux cônes mutuellement polaires*, Comptes Rendus de l'Académie des Sciences de Paris, 255 (1962), pp. 238–240.

[27] T. S. Motzkin, H. Raiffa, G. Thompson, and R. M. Thrall, *The double description method*, in Contributions to Theory of Games, H. Kuhn and A.W.Tucker, eds., vol. 2, Princeton University Press, 1953.

[28] B. A. Murtagh and M. A. Saunders, *MINOS 5.5 user's guide*, Tech. Rep. SOL 83–20R, Department of Operations Research, Stanford University, Palo Alto, California, 1983. Revised July 1998.

[29] The Mathworks, Inc., *Genetic algorithm and direct search toolbox user's guide, version 2.0*. `http://www.mathworks.com/products/gads/`, January 2005.

[30] W. Yu, *Positive basis and a class of direct search techniques*, Scientia Sinica, Special Issue of Mathematics, 1 (1979), pp. 53–67.

[31] W. Yu and Y. Li, *A direct search method by the local positive basis for linearly constrained optimization*, Chinese Annals of Mathematics, 2 (1981), pp. 139–146.

[32] E. H. Zarantonello, *Projections on convex sets in Hilbert space and spectral theory*, in Contributions to Nonlinear Functional Analysis, E. H. Zarantonello, ed., Academic Press, London and New York, 1971, pp. 237–424.