An Active-Set Algorithm for Nonlinear Programming Using Parametric Linear Programming

Richard H. Byrd* Richard A. Waltz†

Revised September 2, 2007

Technical Report, 09/2007

Abstract

This paper describes an active-set algorithm for nonlinear programming that solves a parametric linear programming subproblem at each iteration to generate an estimate of the active set. A step is then computed by solving an equality constrained quadratic program based on this active-set estimate. This approach respresents an extension of the standard sequential linear-quadratic programming (SLQP) algorithm. It can also be viewed as an attempt to implement a generalization of the gradient projection algorithm for nonlinear programming. To this effect, we explore the relation between the parametric method and the gradient projection method in the bound constrained case. Numerical results compare the performance of this algorithm with SLQP and gradient projection.

*Department of Computer Science, University of Colorado, Boulder, CO 80309, USA; richard.byrd@colorado.edu. This author was supported by National Science Foundation grants CCR-0219190, CHE-0205170 and CMMI-0728190, and Army Research Office Grant DAAD19-02-1-0407.

†Department of Industrial & Systems Engineering, University of Southern California, Los Angeles, CA, 90089-0193, USA; rwaltz@usc.edu. This author was supported by National Science Foundation grant CMMI-0728036.

1 Introduction

Although interior-point methods for nonlinear programming have become popular for their ability to efficiently solve large-scale problems with many inequality constraints, active-set methods still constitute an important tool for solving nonlinear optimization problems. Active-set methods are often able to provide more exact solutions and sensitivity information with a clear identification of the active constraint set compared to interior-point methods. Moreover, active-set methods can more effectively make use of a good starting point in a "warm start" procedure (for example, when solving a sequence of closely related problems). In addition, active-set methods are sometimes more stable than interior-point methods as they seem less sensitive to the choice of the initial point and to the scaling of the problem.

For these reasons, active-set methods remain popular and there has been a considerable ongoing effort to improve them [3, 7, 9, 10, 13, 15, 20, 23]. The traditional sequential quadratic programming (SQP) method (e.g., [1, 13, 16, 17]) has proved to be robust and efficient provided the number of degrees of freedom in the problem is not too large. However, efficient implementations of contemporary SQP methods require one to form and factorize a reduced Hessian matrix at each outer iteration. Even if the Hessian matrix is sparse, the reduced Hessian will, in general, not be sparse, making this approach often prohibitively expensive for problems with more than a couple thousand degrees of freedom.

In an attempt to overcome some of the bottlenecks of the traditional SQP approach, there has been a resurgence in recent years in the so-called "EQP" form of sequential quadratic programming. Instead of solving quadratic programming subproblems with inequality constraints, these methods first use some auxiliary mechanism to generate an estimate of the active set, and then solve an *equality* constrained quadratic program (QP) at each iteration, imposing the constraints in the active-set estimate as equalities and temporarily ignoring the other constraints. The sequential linear-quadratic programming (SLQP) algorithm, first proposed by Fletcher etal. [11] is one such method. This approach estimates the active set at each iteration by solving a linear program (LP) based on a linearization of the nonlinear program at the current iterate.

In this paper, we propose an extension to the SLQP approach that solves a sequence of linear programs, instead of a single linear program, to estimate the active set at each iteration. This *parametric* linear programming approach allows us to introduce some quadratic information into the active-set identification phase in the hopes of generating a better active-set estimate. We will refer to this new method as parametric SLQP (or pSLQP for short) to distinguish it from the standard SLQP algorithm.

The paper is organized as follows. In section 2 we provide a brief overview of the standard SLQP approach and discuss some of the drawbacks of this approach. We then introduce our new parametric approach for active-set identification in section 3. In section 4 we discuss the relationship between this new approach and the gradient projection algorithm. Next, in section 5 we describe the details of our specific pSLQP implementation. We provide numerical results that compare our implementation of the parametric SLQP approach with the standard SLQP approach and gradient projection in section 6 and close with some final remarks in section 7.

2 Overview of the SLQP Approach

The SLQP algorithm was first proposed by Fletcher and Sainz de la Maza [11]. More recently Chin and Fletcher proposed an SLQP algorithm based on a filter step acceptance mechanism [6] and Byrd etal [3, 4], proposed a trust-region based SLQP method using an ℓ_1 penalty approach.

In contrast to traditional sequential quadratic programming, which solves a general QP with inequality constraints at each iteration, the SLQP approach first solves an LP to estimate the active set and then solves an equality constrained quadratic program (EQP) to generate a step. An appealing feature of this approach is that it avoids the need to form the reduced Hessian matrix and only requires the solution of LP and EQP subproblems, both of which can be solved efficiently in the large-scale case.

Let us write the nonlinear program (NLP) we wish to solve as

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) && \text{(2.1a)} \\ \text{subject to} \quad & h_i(x) = 0, \quad i \in \mathcal{E} && \text{(2.1b)} \\ & g_i(x) \geq 0, \quad i \in \mathcal{I}, && \text{(2.1c)} \end{aligned}$$

where the objective function $f : I\!R^n \to I\!R$, and the constraint functions $h_i : I\!R^n \to I\!R,\ i \in \mathcal{E}$ $g_i : I\!R^n \to I\!R,\ i \in \mathcal{I}$, are assumed to be twice continuously differentiable.

The SLQP approach will generate an active-set estimate at each iteration k based on solving an LP of the following form,

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \nabla f(x_k)^T d && \text{(2.2a)} \\ \text{subject to} \quad & h_i(x_k) + \nabla h_i(x_k)^T d = 0, \quad i \in \mathcal{E} && \text{(2.2b)} \\ & g_i(x_k) + \nabla g_i(x_k)^T d \geq 0, \quad i \in \mathcal{I} && \text{(2.2c)} \\ & \|d\|_\infty \leq \Delta_k^{\text{LP}}. && \text{(2.2d)} \end{aligned}$$

Here Δ_k^{LP} is an (infinity norm) trust-region radius that ensures the LP is bounded, and that is chosen in a way to try to obtain a good active-set estimate. This parameter plays an important role in the algorithm as its choice at each iteration greatly effects the quality of the active-set estimate. The LP (2.2) is formed by taking a linear approximation of the objective function (2.1a) and constraints (2.1b)-(2.1c) at the current iterate x_k. The linearized problem constraints (2.2b)-(2.2c) active at the solution of this subproblem are used to form the active-set estimate $\mathcal{W}$, which we refer to as the *working set*. Because of the trust-region constraint (2.2d), this subproblem may be infeasible. Therefore we use an ℓ_1 penalty approach to relax the problem constraints (2.2b)-(2.2c). We define a piecewise, linear model

$$\begin{aligned} \ell_k(d) \quad = \quad & f(x_k) + \nabla f(x_k)^T d + \nu_k \sum_{i \in \mathcal{E}} |h_i(x_k) + \nabla h_i(x_k)^T d| \\ & + \nu_k \sum_{i \in \mathcal{I}} \max(0, -g_i(x_k) - \nabla g_i(x_k)^T d), \end{aligned} \tag{2.3}$$

which approximates the (ℓ_1) merit function,

$$\phi(x_k;\nu_k) = f(x_k) + \nu_k \sum_{i\in\mathcal{E}} |h_i(x_k)| + \nu_k \sum_{i\in\mathcal{I}} (\max(0, -g_i(x_k)), \tag{2.4}$$

where $\nu_k > 0$ is a penalty parameter. We then solve the subproblem

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \ell_k(d) && \text{(2.5a)} \\ \text{subject to} \quad & \|d\|_\infty \leq \Delta_k^{\text{LP}}, && \text{(2.5b)} \end{aligned}$$

which is always feasible. Although (2.5) is non-differentiable, it can easily be reformulated and solved as a smooth LP by adding auxiliary variables. A procedure for controlling the penalty parameter is discussed in [3, 4] and is not important to the topic of this paper.

After solving a linear program to generate a working set $\mathcal{W}_k$, a trial step is then generated by solving the equality constrained QP

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \tfrac{1}{2} d^T H(x_k, \lambda_k) d + \nabla f(x_k)^T d && \text{(2.6a)} \\ \text{subject to} \quad & h_i(x_k) + \nabla h_i(x_k)^T d = 0, \quad i \in \mathcal{E} \cap \mathcal{W}_k && \text{(2.6b)} \\ & g_i(x_k) + \nabla g_i(x_k)^T d = 0, \quad i \in \mathcal{I} \cap \mathcal{W}_k && \text{(2.6c)} \\ & \|d\|_2 \leq \Delta_k, && \text{(2.6d)} \end{aligned}$$

where λ_k denotes the vector of Lagrange multiplier estimates and $H(x_k, \lambda_k)$ the Hessian of the Lagrangian of the NLP (2.1). The trust-region constraint (2.6d) is used to handle directions of negative curvature, which may result if $H(x_k, \lambda_k)$ is not positive-definite in the reduced space defined by the constraints (2.6b)-(2.6c). This trust region operates independently of the LP trust-region constraint (2.2d), and is used to ensure descent of the merit function $\phi(x;\nu_k)$.

Although implementations of this standard SLQP approach have provided some encouraging results [3], one significant drawback of this approach is that only linear information is used in identifying the active set via the LP subproblem (2.5). As a result, many trust-region constraints (2.2d) will typically be active at the solution of the LP, and the efficiency of the algorithm and its ability to quickly identify the optimal active set is heavily dependent on the choice of Δ_k^{LP} at each iteration. Although heuristics have been employed for updating Δ_k^{LP} that work reasonably well in practice [3], in general it is not clear how best to choose this parameter at each iteration. Moreover, for highly nonlinear models, the use of only linear information in the active-set identification phase sometimes leads to inefficiencies and inaccurate active-set estimates.

This has motivated us to look at a new mechanism for estimating the active set at each iteration that is based on solving a series of closely related linear programs. This method is inspired by contemporary gradient projection methods for bound constrained optimization, which estimate the active set by minimizing a quadratic model function along a projected gradient path. Similarly, our new approach allows us to incorporate some quadratic information in the active-set identification phase so as to choose our working set in a more clever way.

3 A Parametric LP Approach for Active Set Identification

To motivate our new approach, we start with an example that highlights the difficulty of choosing Δ^{LP} to achieve a good active-set estimate.

Example 1

Consider the example illustrated in Figure 1 below. The elliptical countours are the

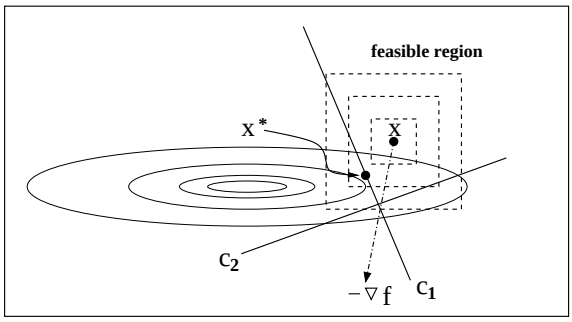

Figure 1: LP active-set estimate for $\Delta^{\text{LP}} = 1, 2, 3$.

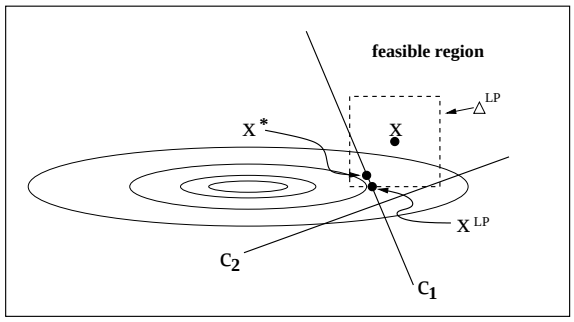

Figure 2: LP solution for $\Delta^{\text{LP}} = 2$.

contours of the objective function we are trying to minimize and the solution point is given by x^*. There are two linear inequality constraints c_1 and c_2 whose feasible region is the area above c_2 and to the right of c_1 as indicated. The dashed boxes represent three different values for the LP trust-region radius, $\Delta^{\text{LP}} = \{1, 2, 3\}$, and the current point is given by x.

From Figure 1 we can see that the only constraint active at the solution is c_1. For $\Delta^{\text{LP}} \geq 3$ the LP solution point x^{LP} will lie at the intersection of c_1 and c_2 and incorrectly identify both of these constraints as active. For $\Delta^{\text{LP}} \leq 1$, both constraints lie outside the LP trust region, and hence no constraints are identified as active. To identify the correct active set requires that we choose Δ^{LP} large enough to include c_1, but small enough to exclude c_2 in the LP solution. For example, as shown in Figure 2, it is easy to see that for $\Delta^{\text{LP}} = 2$, the solution, x^{LP}, of the linear program identifies the correct active constraint c_1.

In general, however, it is not known how to choose Δ^{LP} in the required range to identify the optimal active set near the solution. Moreover, the range of values that identify the optimal active set may be quite small making a good choice for Δ^{LP} unlikely. To address this difficulty, instead of solving a single linear program with a fixed trust region Δ^{LP} to determine the working set at each iteration, we propose to solve instead a series of linear programs that will give us the LP solution points for a range of Δ^{LP} values. We will then approximately minimize a quadratic model along the path generated from this set of parametric solutions, to determine a Cauchy point and our working set.

More precisely, we propose the following mechanism for determining a working constraint set $\mathcal{W}_k$ at each iteration as well as a Cauchy step d_k^{C} that provides sufficient reduction in a model of our merit function and that can be used to establish global convergence. To facilitate this we define a piecewise quadratic-linear model of the merit function which was defined in (2.4):

$$q_k(d) = \tfrac{1}{2} d^T H(x_k, \lambda_k) d + \ell_k(d). \tag{3.7}$$

We assume here that the value of the penalty parameter ν_k has already been set to an appropriate level and will remain fixed for the remainder of the given iteration. We then determine our estimate of the active constraint set at each iteration as follows:

1. Use a parametric LP solve to obtain the solutions of (2.5) for all values of $\Delta^{\rm LP} \in [\Delta^{\rm min}, \Delta^{\rm max}]$. Denote these parameterized LP solutions as $d_k^{\rm LP}(\Delta^{\rm LP})$, and let the path defined by this set of solutions be $\mathcal{P}_k(\Delta^{\rm LP})$.

2. Determine the optimal trust-region size $\Delta_k^{\rm LP}$ as the solution of

$$\min q_k(d_k^{\rm LP}(\Delta^{\rm LP})), \quad \Delta^{\rm LP} \in [\Delta^{\rm min}, \Delta^{\rm max}], \tag{3.8}$$

   and define the Cauchy step to be $d_k^{\rm C} = d_k^{\rm LP}(\Delta_k^{\rm LP})$.

3. Define the working set $\mathcal{W}_k$ to be a linearly independent subset of the active linearized constraints (2.2b)-(2.2c) at $d_k^{\rm C}$.

Referring to Example 1, it is easy to see that the step $d_k^{\rm C}$ generated by the parametric solution procedure above will identify the correct active set $\{c_1\}$, provided the searching range $[\Delta^{\rm min}, \Delta^{\rm max}]$ is chosen large enough to overlap the optimal active-set identification range for $\Delta^{\rm LP}$.

We now summarize our new algorithm using the pseudo-code of Algorithm 3.1 below.

Algorithm 3.1: Parametric SLQP Algorithm

Initial data: x_0, $\Delta_0 > 0$, $\Delta^{\rm max} > \Delta^{\rm min} > 0$, $0 < \rho_u \leq \rho_s < 1$.

For $k = 0, 1, \ldots$, until a stopping test is satisfied, perform the following steps.

1. Parametric path Generate the path $\mathcal{P}_k(\Delta^{\rm LP})$ via a parametric solution of the LP (2.5) for $\Delta^{\rm LP} \in [\Delta^{\rm min}, \Delta^{\rm max}]$.

2. Working set Minimize a model function $q_k(d)$ along the parametric solution path $\mathcal{P}_k(\Delta^{\rm LP})$ to obtain a Cauchy step $d_k^{\rm C}$ and working set $\mathcal{W}_k$.

3. Trial point Compute a trial point $x_k^{\rm T} = x_k + d_k$ constructed from the step obtained by (approximately) solving the equality constrained quadratic program (2.6). Enforce that the trial step d_k satisfies $q_k(d_k) \leq q_k(d_k^{\rm C})$.

4. Compute

$$\rho_k = \frac{\phi(x_k; \nu_k) - \phi(x_k^{\rm T}; \nu_k)}{q_k(0) - q_k(d_k)}.$$

5a. If $\rho_k \geq \rho_s$, choose $\Delta_{k+1} \geq \Delta_k$, otherwise choose $\Delta_{k+1} < \Delta_k$.

5b. If $\rho_k \geq \rho_u$, set $x_{k+1} \leftarrow x_k^{\rm T}$, otherwise set $x_{k+1} \leftarrow x_k$.

Many details of Algorithm 3.1 have been left purposely vague as the primary aim of this paper is to focus on the active-set identification mechanism. Details such as the EQP step computation, the trust-region update rules, the Lagrange multiplier computations and the penalty parameter update are described in [3] and will not be repeated here. The definition of the Cauchy point d_k^{C} and the requirement in Step 3 of Algorithm 3.1, that our final trial step provide at least as much reduction in our model function $q_k(d_k)$ as this Cauchy point provides favorable global convergence properties for this algorithm. Global convergence using a slightly different Cauchy step for a related SLQP algorithm was established in [4]. In section 5, we will discuss in more depth the trial point computation.

The important point to note, is that Algorithm 3.1 (in contrast to standard SLQP algorithms) explores a range of possible working sets for different values of Δ^{LP} and incorporates quadratic information in the active-set identification phase of the algorithm via the model function $q_k(d)$ in Step 2. This procedure of minimizing a model function with quadratic information along a path generated from linear information is similar in flavor to contemporary gradient projection methods [21]. Indeed, a motivation for this approach is to define an algorithm like gradient projection, but one that applies more readily to problems with general inequality constraints. We now in fact explore the relation between the two methods.

4 Parametric SLQP versus Gradient Projection

In the gradient projection algorithm a path is defined by projecting scalar multiples of the negative gradient vector onto the feasible set. Then a quadratic model function can be minimized along that path to determine a working set. This piecewise path is different from the piecewise path $\mathcal{P}_k(\Delta^{\mathrm{LP}})$ that results from the parametric solution of (2.5), even in the case of bound constraints. However, if we generalize the gradient projection approach to gradients in other norms we can see that, for bound constraints, gradient projection and pSLQP coincide in the case of the sup norm gradient. To be precise, if we define the gradient, or steepest descent direction with respect to a given norm $\|\cdot\|_p$ (see [14]) as the solution d^{SD_p} to the problem

$$\underset{d}{\text{minimize}} \quad \nabla f(x)^T d \tag{4.9}$$

$$\text{subject to} \quad \|d\|_p \leq \|\nabla f(x)\|_p, \tag{4.10}$$

then in the case of the ℓ_∞ norm, each component, i, of the steepest descent direction is given by

$$d_i^{\mathrm{SD}_\infty} = -\mathrm{sign}(\nabla f(x)_i)\|\nabla f(x)\|_\infty. \tag{4.11}$$

Below we establish a formal equivalence in the simple bound constrained case between the pSLQP approach and gradient projection in this norm.

Consider the simple bound constrained LP with trust region constraint Δ^{LP}:

$$\underset{d}{\text{minimize}} \quad \nabla f(x)^T d \tag{4.12a}$$

$$\text{subject to} \quad u_i \leq d_i \leq v_i, \quad \forall i \tag{4.12b}$$

$$\|d\|_\infty \leq \Delta^{\mathrm{LP}}, \tag{4.12c}$$

where $u_i \leq 0 \leq v_i$, $\forall i$. This problem can be written more compactly as:

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \nabla f(x)^T d && \text{(4.13a)} \\ \text{subject to} \quad & \max(u_i, -\Delta^{\text{LP}}) \leq d_i \leq \min(v_i, \Delta^{\text{LP}}), \quad \forall i. && \text{(4.13b)} \end{aligned}$$

The following theorem establishes an equivalence between the solutions of (4.13) parameterized by Δ^{LP}, and the gradient projection path defined in the infinity norm.

Theorem 4.1 *Define the set $B = \{d \mid u_i \leq d_i \leq v_i,\ i = 1..n\}$, and denote the projection of $\tau d^{SD\infty}$ for $\tau > 0$ onto B in some norm as*

$$P_B(\tau d^{SD\infty}) = argmin_d \|\tau d^{SD\infty} - d\| \text{ subject to } d \in B. \tag{4.14}$$

Then for any $\tau > 0$ the projection in any norm of the steepest descent gradient $d^{GP}(\tau) = P_B(\tau d^{SD\infty})$, given by (4.11), is a solution $d^{LP}(\Delta^{LP})$ to the parameteric LP (4.13). If the ℓ_2 projection is used then $d^{LP}(\Delta^{LP})$ is the unique projection.

Proof. Clearly all solutions of (4.13) are characterized as follows:

$$d_i^{\text{LP}}(\Delta^{\text{LP}}) = \begin{cases} \max(u_i, -\Delta^{\text{LP}}) & \text{if } \sigma_i > 0, \\ \min(v_i, \Delta^{\text{LP}}) & \text{if } \sigma_i < 0, \\ \text{any value in } [\max(u_i, -\Delta^{\text{LP}}), \min(v_i, \Delta^{\text{LP}})] & \text{if } \sigma_i = 0, \end{cases} \tag{4.15}$$

where $\sigma_i = \text{sign}(\nabla f(x)_i)$. For some norms the projections of $\tau d^{\text{SD}\infty}$ onto B are unique, but in all cases the vector whose $i-$th component is the projection of $\tau d_i^{\text{SD}\infty}$ onto the interval $[u_i, v_i]$ is a valid projection of $\tau d^{\text{SD}\infty}$. If we set $\tau = \Delta^{\text{LP}}/\|\nabla f(x)\|_\infty$, then the $i-$th component of $\tau d^{\text{SD}\infty}$ is $-\sigma_i \Delta^{\text{LP}}$ and its projection onto $[u_i, v_i]$, which is given by

$$d_i^{\text{GP}}(\tau) = \begin{cases} u_i & \text{if } -\sigma_i\Delta^{\text{LP}} < u_i, \\ v_i & \text{if } -\sigma_i\Delta^{\text{LP}} > v_i, \\ -\sigma_i\Delta^{\text{LP}} & \text{if } u_i \leq -\sigma_i\Delta^{\text{LP}} \leq v_i, \end{cases} = \begin{cases} \max(u_i, -\Delta^{\text{LP}}) & \text{if } \sigma_i > 0, \\ \min(v_i, \Delta^{\text{LP}}) & \text{if } \sigma_i < 0, \\ 0 & \text{if } \sigma_i = 0, \end{cases} \tag{4.16}$$

is a valid projection of $\tau d^{\text{SD}\infty}$ onto B. Thus, comparing the right-hand side of (4.16) and (4.15), we see that the projection $P_B(\tau d^{\text{SD}\infty})$ is a solution to (4.13) if $\|\tau d^{\text{SD}\infty}\|_\infty = \Delta^{\text{LP}}$. □

This equivalence gives us reason to hope that the performance of parametric SLQP will approach that of projected gradient. When we move beyond bound constrained problems to more general constraints this equivalence does not hold; in fact gradient projection is very expensive to implement while parametric SLQP remains computationally practical. That fact is a major motivation for this study.

5 Implementation Details

Algorithm 3.1 describes a general outline for our parametric SLQP approach. However, to test the practical performance of this type of algorithm requires a specific software implementation where many details of this algorithm are refined. Below we describe some aspects of the software implementation that are not fully explained by Algorithm 3.1, or that deviate from this idealized algorithm. This software implementation is used to generate the numerical results in the section that follows.

5.1 Approximate parametric solve

In practice, and for the numerical results described in the next section, instead of exactly solving a parametric LP, we will approximate this by solving (2.5) for a finite number of Δ^{LP} trial values. This approximation is used because of the lack of available software for completely solving parametric LPs and also for efficiency.

Algorithm 5.1: Approximate Parametric LP Solve

Initial data: x_k, $\Delta^{\mathrm{LP}_0} > 0$, *done* =`FALSE`, $j = 0$, $j^{\max} > 0$.

Solve the LP (2.5) with trust region Δ^{LP_j} to obtain step d^{LP_j}, and $\mathcal{W}^j$.
Choose ν_k; compute λ_k and $H(x_k, \lambda_k)$.
If $q_k(d^{\mathrm{LP}_j}) > q_k(0) - 0.1[\ell_k(0) - \ell_k(d^{\mathrm{LP}_j})]$ Then
    Set *searchDir* = `backtrack`, $\Delta^{\mathrm{LP}_{j+1}} = 0.5\Delta^{\mathrm{LP}_j}$.
Else
    Set *searchDir* = `forward_search`, $\Delta^{\mathrm{LP}_{j+1}} = 2\Delta^{\mathrm{LP}_j}$.
Endif
While *done* ==`FALSE` and $j < j^{\max}$
    Set $j \leftarrow j + 1$.
    Solve the LP (2.5) with trust region Δ^{LP_j} to obtain step d^{LP_j}, and $\mathcal{W}^j$.
    If *searchDir* == `backtrack` Then
        If $q_k(d^{\mathrm{LP}_j}) > q_k(0) - 0.1[\ell_k(0) - \ell_k(d^{\mathrm{LP}_j})]$ Then
            Set $\Delta^{\mathrm{LP}_{j+1}} = 0.5\Delta^{\mathrm{LP}_j}$.
        Else
            Set *done* =`TRUE`.
        Endif
    Elseif *searchDir* == `forward_search` Then
        If $q_k(d^{\mathrm{LP}_j}) < q_k(d^{\mathrm{LP}_{j-1}})$ Then
            Set $\Delta^{\mathrm{LP}_{j+1}} = 2.0\Delta^{\mathrm{LP}_j}$.
        Else
            Set $d^{\mathrm{LP}_j} \leftarrow d^{\mathrm{LP}_{j-1}}$, $\mathcal{W}^j \leftarrow \mathcal{W}^{j-1}$, $\Delta^{\mathrm{LP}_j} \leftarrow \Delta^{\mathrm{LP}_{j-1}}$.
            Set *done* =`TRUE`.
        Endif
EndWhile
Set $\mathcal{W}_k = \mathcal{W}^j$, $\Delta_k^{\mathrm{LP}} = \Delta^{\mathrm{LP}_j}$.
If $j = j^{\max}$ and *searchDir* == `backtrack` Then
    Set $d_k^{\mathrm{C}} = \alpha^{\mathrm{LP}} d^{\mathrm{LP}_j}$ where α^{LP} solves $\min q_k(\alpha d^{\mathrm{LP}_j}), \alpha \in [0, 1]$.
Else
    Set $d_k^{\mathrm{C}} = d^{\mathrm{LP}_j}$.
Endif

Algorithm 5.1, describes in more detail how we compute this approximate solution. The general idea is to solve for an initial value of Δ^{LP}. If this initial solve does not provide a sufficient reduction in the model $q_k(d)$ of our merit function, then we will choose a sequence

of increasingly smaller values of Δ^{LP} until a sufficient reduction is obtained. Otherwise, if sufficient reduction is achieved with the initial solve, then we will choose an increasing sequence of Δ^{LP} values until the sufficient reduction condition no longer holds.

The superscript indices in Algorithm 5.1 indicate iterations within the approximate parametric solve subproblem as opposed to outer algorithmic iterations, which are identified using subscripts. In the numerical results of the next section we use $j^{\max} = 5$. Although this approach may require up to $j^{\max}$ LP solves per iteration, these LP solves can be effectively warm started using a simplex solver so that the additional LP solves beyond the initial one are solved very quickly.

The value of the penalty parameter ν_k in Algorithm 5.1 is chosen based on the initial trust-region value, Δ^{LP0}, according to the update rules described in [4]. Although the final LP trust region Δ_k^{LP} will often differ from the initial guess Δ^{LP0}, we believe the procedure described in [4] for setting ν_k is still appropriate in this case and that the parametric procedure does not adversly affect our penalty update strategy.

Since our parametric solve is only approximate, at the end of Algorithm 5.1 if d^{LP} is halved $j^{\max}$ times, to save computational time at this stage we perform a simple linesearch along the last LP step to determine the Cauchy step d_k^{C}.

5.2 Trust-region initialization for the parametric LP

Although the rule for choosing Δ_k^{LP} at each iteration is crucial to the efficiency of the standard SLQP algorithm, which only solves one LP per iteration, the initial guess of this parameter Δ^{LP0} is much less important in our parametric approach since we explore a range of Δ^{LP} values each iteration. Indeed, this was a primary motivation for proposing the parametric approach. Nonetheless, we attempt to choose a good value of Δ^{LP0} to initialize our approximate parametric LP solve described above, so as to improve the accuracy and efficiency of this solve.

We initialize Δ^{LP0} at the beginning of the approximate parametric solve as follows. If the trial step d_k taken by the algorithm on the most current iteration was accepted we define

$$\Delta_{k+1}^{\text{LP0}} = \begin{cases} \min(\max\{1.2\|d_k\|_\infty, 1.2\|d_k^{\text{C}}\|_\infty, 0.1\Delta_k^{\text{LP}}\}, 7\Delta_k^{\text{LP}}), & \text{if } \alpha_k^{\text{LP}} = 1 \\ \min(\max\{1.2\|d_k\|_\infty, 1.2\|d_k^{\text{C}}\|_\infty, 0.1\Delta_k^{\text{LP}}\}, \Delta_k^{\text{LP}}), & \text{if } \alpha_k^{\text{LP}} < 1 \end{cases}, \tag{5.17}$$

whereas if the step d_k was rejected we set

$$\Delta_{k+1}^{\text{LP0}} = \min(\max\{0.5\|d_k\|_\infty, 0.1\Delta_k^{\text{LP}}\}, \Delta_k^{\text{LP}}). \tag{5.18}$$

The motivation for (5.17) is to estimate the "optimal" trust region for the new LP subproblem to be a little larger than the norm of the most recent step. In addition bounds are provided to limit the new estimate from increasing or decreasing too much compared with the final trust-region value used in the previous iteration. The initialization (5.18) estimates the optimal LP trust-region to be a fraction of the most recent trial step when this step is rejected. Regardless, of this initial estimate, the parametric approach will try multiple values of Δ^{LP} determined by the parameter $j^{\max}$ in Algorithm 5.1.

5.3 Computing a trial step

In computing a trial step d_k, in our pSLQP algorithm there are two aims:

1. Ensure that the trial point $x_k^{\mathrm{T}} = x_k + d_k$ lies on the subspace defined by our current working set estimate $\mathcal{W}_k$.

2. Make sure that the reduction in the model function $q_k(d)$ achieved by our trial step is at least as good as the reduction obtained by the Cauchy step (i.e. choose d_k such that $q_k(d_k) \leq q_k(d_k^{\mathrm{C}})$).

The first condition is a common condition of active-set methods. The second condition is essential in developing a globally convergent algorithm (see for instance [4]). It is easy to see that these two conditions can always be satisfied in Algorithm 3.1 by defining $d_k = d_k^{\mathrm{C}}$. However, for fast convergence, it is important to use the EQP phase to try to compute a trial step that is better than the Cauchy step in reducing our model function. In our algorithm we do this by defining our trial step to be

$$d_k = d_k^{\mathrm{C}} + \alpha_k^{\mathrm{EQP}} d_k^{\mathrm{CE}}, \tag{5.19}$$

where $d_k^{\mathrm{CE}} = d_k^{\mathrm{EQP}} - d_k^{\mathrm{C}}$ is the line segment leading from the Cauchy point to the EQP solution point and α_k^{EQP} is the approximate solution of

$$\underset{\alpha \in [0\ 1]}{\text{minimize}} \quad q_k(d_k^{\mathrm{C}} + \alpha d_k^{\mathrm{CE}}). \tag{5.20}$$

Although this often works well, it has been observed in practice, that if the active-set estimate is poor, the step d_k^{CE} leading from the Cauchy point to the EQP solution may quickly violate one or more inactive constraints not in the working set (and, hence, not imposed as constraints in the EQP). This may result in the step being severely truncated to satisfy the violated inactive constraints, leading to steepest descent-like steps and slow convergence. In particular, this behavior is prevalent in problems with degenerate constraints where, in order to maintain linear independence, there may exist active constraints that are not included in the working set. An attempt to overcome this problem, may be made by modifying the algorithm so as to project the step d_k^{CE} onto the feasible region or perhaps allow the algorithm to add inactive constraints encountered during the EQP to the working set. The challenge is to do this in an efficient way and in a way that maintains a linearly independent active set.

For problems whose only constraints are simple bounds, projecting the step d_k^{CE} onto the feasible region is a trivial computation. For the results presented in the next section, we experiment with such a procedure on bound constrained problems to measure what affect this has on speeding up the active-set identification. In particular, for bound constrained problems, we will project the step d_k^{CE} onto the feasible space and perform a linesearch along the piecewise linear path resulting from this projection. In this case the trial step is defined as

$$d_k = P(d_k^{\mathrm{C}} + \alpha_k^{\mathrm{EQP}} d_k^{\mathrm{CE}}), \tag{5.21}$$

where $P(d)$ represents the minimum norm projection of d onto the feasible region and α_k^{EQP} is the approximate solution of

$$\underset{\alpha \in [0\ 1]}{\text{minimize}} \quad q_k(P(d_k^{\text{C}} + \alpha d_k^{\text{CE}})). \tag{5.22}$$

In the case when there are more general constraints, the projection of d_k^{CE} onto the feasible space is a more expensive computation. As a result it is not clear whether such a feature can be efficiently used for more general problems. We feel it is worth highlighting this feature as it leads to a significant improvement in performance on some problems whose only constraints are simple bounds and works particularly well when used in conjunction with the parametric LP active-set identification procedure.

6 Numerical Tests

In this section we compare the performance of the SLQP approach with the parametric SLQP (pSLQP) approach described in the previous section. Both approaches have been implemented in the KNITRO software package version 5.1 [5, 25]. On bound constrained problems we also compare with the gradient projection code TRON version 1.2 [18]. All tests were run on a dedicated 32-bit Intel Pentium IV machine with 1Gb of RAM running Redhat Fedora Core 4.

6.1 Bound Constrained Problems

First we want to restrict our attention to problems with only simple bound constraints on the variables. By looking at this class of problems in isolation we can compare the active-set identification properties of the SLQP and pSLQP approaches with the performance of the gradient projection approach.

As already mentioned, the parametric LP approach may be viewed as a generalizaton of the gradient projection approach to polyhedral sets. Therefore, it is natural to compare our parametric SLQP approach with the gradient projection method on bound constrained problems, where the latter method has proved to be quite efficient. We are interested in the two approaches as active set predictors, and thus a relevant performance measure is number of function evaluations.

In our tests we compare a version of SLQP and pSLQP implemented in the KNITRO software package [5, 25], with the gradient projection code TRON [18]. The LANCELOT [7], and LBFGS [26] solvers also implement effective gradient projection algorithms for bound constrained problems. We choose to compare with TRON since it is widely regarded as one of the most efficient codes for bound constrained optimization, whose implementation – apart from the active set estimation scheme – is very similar in many ways to our parametric SLQP implementation. Both codes are trust-region methods that use exact second derivative information and solve the equality constrained subproblems using a conjugate gradient approach. We use the default stopping tests in both KNITRO and TRON, which we find in most cases generates solutions with similar accuracy. We have also confirmed that all algorithms converge to the same local solution.

Our test set consists of bound constrained problems in the CUTEr test set [2] that have been translated to Ampl models [12]. These Ampl versions of the CUTEr test problems are available from [24]. We exclude from our bound constrained test set all the models for which the default number of variables is less than 1000. The remaining 31 bound constrained problems form our test set. All the results in this subsection will be presented using the performance profiles proposed by Dolan and Moré [8]. In the plots $\pi_s(\tau)$ denotes the logarithmic performance profile

$$\pi_s(\tau) = \frac{\text{no. of problems where} \log_2(r_{p,s}) \leq \tau}{\text{total no. of problems}}, \qquad \tau \geq 0, \tag{6.23}$$

where $r_{p,s}$ is the ratio between the number of function evaluations to solve problem p by solver s over the fewest number of function evaluations required by any of the solvers. See [8] for more details regarding performance profiles.

As noted in section 5.3, slow convergence of our algorithm may result if inactive bounds are violated during the solution of the EQP. It was observed that TRON has a feature that allows it to cope with this problem by adding constraints during the step computation phase if inactive bounds are violated (and then re-solving the EQP). On some problems, it has been observed that this feature significantly improves the performance of the algorithm.

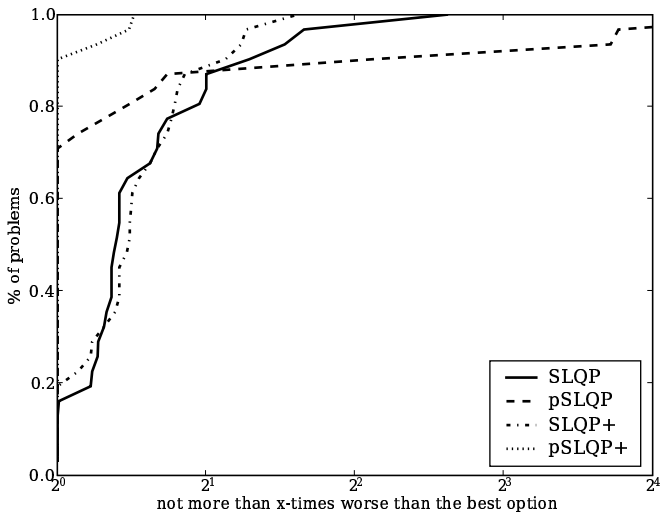

Figure 3: Comparing SLQP, pSLQP, SLQP+ and pSLQP+ in terms of functions evaluation counts on 31 bound constrained problems.

To make a more meaningful comparison with TRON of active set strategies on bound constrained problems, we have added a similar feature to our SLQP and pSLQP algorithms. We do not explicitly add violated constraints during the unconstrained minimization as TRON does (since this may significantly increase the cost per iteration). Rather, we project the step onto the feasible space if constraints are violated during this minimization and then perform a linesearch along this projected path as described by (5.21) and (5.22) in

section 5.3. We refer to these variations of our algorithms as SLQP+ and pSLQP+. We implement this projected EQP step strategy only for bound constrained problems since the extension of this technique (like extending the TRON approach) to more general constraints is much more complemented and computationally expensive.

In Figure 3, we present results comparing the SLQP+ and pSLQP+ variants with the standard SLQP and pSLQP algorithms, which truncate the step as described by (5.19) and (5.20) to satisfy the bounds if necessary. As can be seen from Figure 3, the results for the parametric approach work better than the standard SLQP approach in most cases. In particular, the parametric approach that projects the step onto the feasible space (pSLQP+) is clearly the most efficient variation. This seems to highlight that a procedure for dealing with inactive constraints encountered during the step computation can greatly enhance these types of methods.

In Figure 4 we present results comparing SLQP+, pSLQP+ and TRON on our bound constrained problem test set. Note, in Figure 4, that both TRON and pSLQP+ are signif-

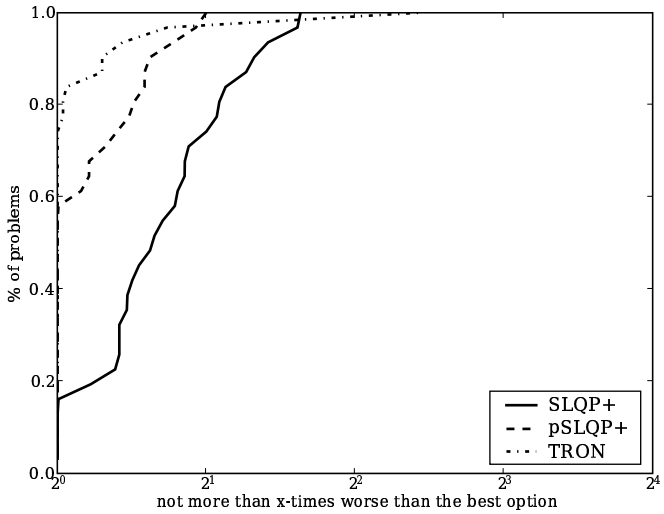

Figure 4: Comparing SLQP+, pSLQP+ and Tron in terms of functions evaluation counts on 31 bound constrained problems.

icantly better than the SLQP+ approach. This indicates that these methods, which have more sophisticated active-set identification procedures incorporating quadratic information, do a better job of predicting the active-set as expected.

All detailed results from this section are reported in Table 1 in the Appendix. We also indicate here the number of variables n in each problem and the number of bounds active at the solution of each problem ($|\mathcal{A}^*|$). In general it is noteworthy how similar the performance is between the pSLQP+ approach and TRON. The algorithm implemented in pSLQP+ is quite similar to TRON differing significantly only in the path used in the active-set estimate (parametric LP path versus gradient projection path) and in the strategy used for bounds encountered in the EQP stage, as described above. These empirical results seem to confirm

the similarities between these approaches established in section 4.

6.2 Quadratic Programming Problems

A primary advantage of the new method over gradient projection is that it can be used to solve problems with general constraints. (In fact, we know of no practical software implementation of gradient projection for general constraints). In this section we compare SLQP versus pSLQP on all the CUTEr test problems translated into Ampl models that are quadratic programming problems. QPs whose only constraints are simple bounds are excluded from this test set to avoid redundancy (i.e., it does not include problems from the previous section). This test set consists of 91 problems. The SLQP algorithm reported finding an optimal solution on 86 of these problems while pSLQP found the optimal solution on 85 of the problems.

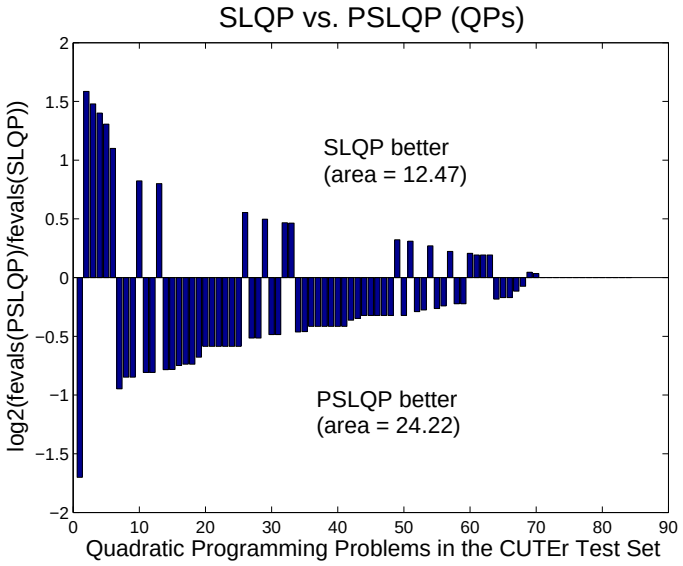

Figure 5: Results on 84 quadratic programming problems comparing SLQP and pSLQP in terms of functions evaluation counts.

In Figure 5 we compare SLQP and pSLQP in terms of function evaluation count using a plotting technique first proposed by Morales [19] for comparing two methods. Figure 5 plots the ratio

$$\log_2 \frac{(\text{fevals pSLQP})}{(\text{fevals SLQP})}, \tag{6.24}$$

where fevals denotes the number of function evaluations required to meet the stopping test. In the figure, the problems are arranged along the horizontal axis in decreasing order of the absolute value of (6.24). A bar in the positive y-axis indicates that the SLQP approach was better (i.e., required fewer function evaluations), while a bar in the negative y-axis indicates that the pSLQP approach was better. We also report the total area for each approach to provide some overall comparative measure of performance. In this comparison, we only look at those 84 problems that both algorithms solved and found the same local solution.

The results indicate that on quadratic programs the pSLQP is overall more efficient than the SLQP approach.

6.3 General Nonlinear Problems

In this section we compare SLQP versus pSLQP on a set of CUTEr test problems translated into Ampl models that are general nonlinear programming problems with at least one inequality constraint or bound. It excludes bound constrained problems and QPs. There are 246 problems in this test set. The SLQP algorithm finds an optimal solution on 213 of these problems while pSLQP found the optimal solution on 211 of the problems.

In Figure 6 we compare SLQP and pSLQP in terms of function evaluation count using the same log plot as in the previous subsection. In this comparison, we only look at those 196 problems that both algorithms solved and found the same local solution. The results for general constrained problems provide only a slight advantage for the pSLQP approach. It is unclear why the pSLQP approach – which seems to perform better than SLQP on bound constrained problems and QPs – does not do better on more general problems. Since we know of no gradient projection implementation for general nonlinear problems, the active-set identification properties of these types of methods in this context are not well understood. We hypothesize that it is more likely in problems with nonlinear constraints, that the EQP step may quickly violate one or more inactive constraints causing the step to be unduly truncated, and that, as a result, these methods require an effective scheme for adding constraints encountered during the EQP phase. We regard it as an interesting topic that requires further research.

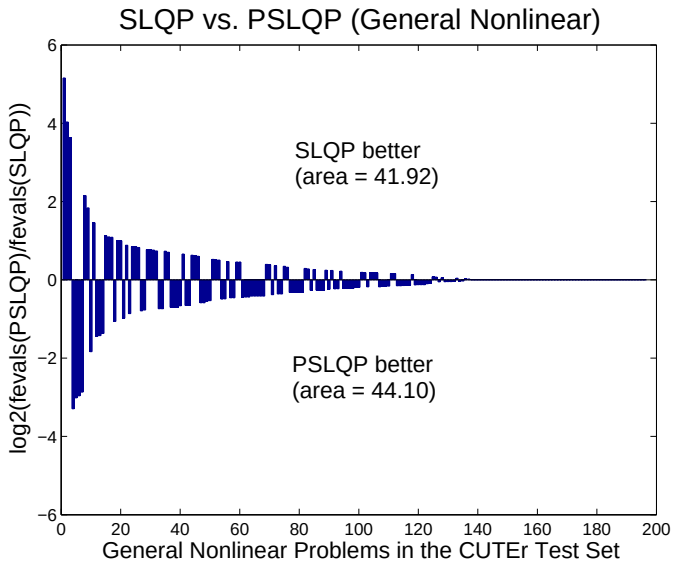

Figure 6: Results on 196 general nonlinear problems comparing SLQP and pSLQP in terms of functions evaluation counts.

7 Final Remarks

We have seen that the parametric SLQP approach presented in this paper is much more efficient than the old approach on bound constrained problems. Therefore, when this method is incorporated into a general package, such as the active-set algorithm in KNITRO, it may no longer be necessary to have specialized packages for bound constraints. We have also observed significant gains for quadratic programs. More work is needed to determine whether this new approach is superior on general constraints.

The work of Oberlin and Wright [22] provides some theory that stipulates under what conditions the LP subproblems solved in SLQP methods identify the correct active set. However, it remains to show, how to develop effective trust-region update rules for SLQP methods, such that the required conditions are automatically satisfied as a solution is approached. A more comprehensive theory for active-set identification in the context of both SLQP and pSQLP methods is a focus of ongoing research.

8 Appendix

Problem	n	$\lvert\mathcal{A}^*\rvert$	# of function evaluations				
			SLQP	SLQP+	pSLQP	pSLQP+	TRON
biggsb1	1000	2	506	503	503	503	501
bqpgauss	2003	95	233	38	500	53	206
chenhark	1000	395	848	633	334	269	204
cvxbqp1	10000	10000	2	2	2	2	2
gridgena	11542	0	7	7	6	6	10
jnlbrng1	15129	4884	36	36	29	29	26
jnlbrng2	15129	5990	31	35	16	16	16
jnlbrnga	15129	5161	29	42	24	24	25
jnlbrngb	15129	6652	22	19	11	11	11
mccormck	50000	1	9	7	11	10	7
minsurfo	2500	784	10	11	10	12	7
ncvxbqp1	10000	10000	4	4	4	4	2
ncvxbqp2	10000	9935	20	21	21	15	10
ncvxbqp3	10000	9831	25	25	25	15	10
nobndtor	14400	2322	49	52	232	17	23
nonscomp	10000	0	16	17	11	10	8
obstclae	15129	7178	66	65	959	27	27
obstclbm	15129	3817	42	38	91	21	21
pentdi	1000	998	2	2	2	2	2
torsion1	14400	4416	47	51	39	39	40
torsion2	14400	4416	52	56	40	40	21
torsion3	14400	9192	27	28	21	21	21
torsion4	14400	9192	29	35	22	22	19
torsion5	14400	11832	16	17	12	12	12

continued on next page

continued from previous page							
Problem	n	$\|\mathcal{A}^*\|$	# of function evaluations				
			SLQP	SLQP+	pSLQP	pSLQP+	TRON
torsion6	14400	11832	18	19	13	13	16
torsiona	14400	4288	49	52	39	39	40
torsionb	14400	4288	47	47	40	40	26
torsionc	14400	9128	27	29	21	21	21
torsiond	14400	9128	35	31	22	22	19
torsione	14400	11800	16	16	12	12	12
torsionf	14400	11800	20	20	13	13	16

Table 1: Comparison of SLQP, SLQP+, pSLQP, pSLQP+ and TRON on bound constrained problems in terms of function evaluations.

Acknowledgments. The authors thank Jorge Nocedal who provided many valuable suggestions and insightful discussions during the course of this work.

References

[1] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.

[2] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[3] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Mathematical Programming, Series B*, 100(1):27–48, 2004.

[4] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. On the convergence of successive linear-quadratic programming algorithms. *SIAM Journal on Optimization*, 16(2):471–489, 2006.

[5] R. H. Byrd, J. Nocedal, and R.A. Waltz. KNITRO: An integrated package for nonlinear optimization. In G. di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, pages 35–59. Springer, 2006.

[6] C. M. Chin and R. Fletcher. On the global convergence of an SLP-filter algorithm that takes EQP steps. *Mathematical Programming, Series A*, 96(1):161–177, 2003.

[7] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for Large-scale Nonlinear Optimization (Release A)*. Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

[8] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming, Series A*, 91:201–213, 2002.

[9] A. Drud. CONOPT – a large scale GRG code. *ORSA Journal on Computing*, 6:207–216, 1994.

[10] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91:239–269, 2002.

[11] R. Fletcher and E. Sainz de la Maza. Nonlinear programming and nonsmooth optimization by successive linear programming. *Mathematical Programming*, 43(3):235–256, 1989.

[12] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Press, 1993. `www.ampl.com`.

[13] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47:99–131, 2005.

[14] M. Golomb and R. A. Tapia. The metric gradient in normed linear spaces. *Numerische Mathematik*, 20:115–124, 1972.

[15] N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD, a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Softw.*, 29(4):353–372, 2003.

[16] N. I. M. Gould, D. Orban, and Ph. L. Toint. Numerical methods for large-scale nonlinear optimization. *Acta Numerica*, 14:299–361, 2005.

[17] N. I. M. Gould and Ph. L. Toint. SQP methods for large-scale nonlinear programming. In M. J. D. Powell and S. Scholtes, editors, *Proceedings of the IFIP TC7 Conference on System Modelling and Optimization, Cambridge*, pages 149–178, Dordrecht, The Netherlands, 2000. Kluwer Academic Publishers.

[18] C. J. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[19] J. L. Morales. A numerical study of limited memory BFGS methods. *Applied Mathematics Letters*, 15(4):481–487, 2002.

[20] B. A. Murtagh and M. A. Saunders. MINOS 5.4 user's guide. Technical report, SOL 83-20R, Systems Optimization Laboratory, Stanford University, 1983. Revised 1995.

[21] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, second edition, 2006.

[22] C. Oberlin and S. J. Wright. Active constraint identification in nonlinear programming. Technical Report 05-01, Computer Sciences Department, University of Wisconsin, Madison, WI, USA, 2005. To appear in SIAM Journal on Optimization.

[23] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82(3):413–448, 1998.

[24] R. J. Vanderbei. AMPL models. http://www.sor.princeton.edu/ rvdb/ampl/nlmodels.

[25] R. A. Waltz and T. D. Plantenga. KNITRO 5.0 User's Manual. Technical report, Ziena Optimization, Inc., Evanston, IL, USA, February 2006.

[26] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 78: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.