An Algorithm for Nonlinear Optimization Using Linear Programming and Equality Constrained Subproblems

Richard H. Byrd* Nicholas I.M. Gould† Jorge Nocedal‡
Richard A. Waltz‡

Revised May 12, 2003

Report OTC 2002/4, Optimization Technology Center

Abstract

This paper describes an active-set algorithm for large-scale nonlinear programming based on the successive linear programming method proposed by Fletcher and Sainz de la Maza [10]. The step computation is performed in two stages. In the first stage a linear program is solved to estimate the active set at the solution. The linear program is obtained by making a linear approximation to the ℓ_1 penalty function inside a trust region. In the second stage, an equality constrained quadratic program (EQP) is solved involving only those constraints that are active at the solution of the linear program. The EQP incorporates a trust-region constraint and is solved (inexactly) by means of a projected conjugate gradient method. Numerical experiments are presented illustrating the performance of the algorithm on the CUTEr [1] test set.

*Department of Computer Science, University of Colorado, Boulder, CO 80309; richard@cs.colorado.edu. This author was supported by Air Force Office of Scientific Research grant F49620-00-1-0162, Army Research Office Grant DAAG55-98-1-0176, and National Science Foundation grant INT-9726199.

†Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0Qx, England, EU; n.gould@rl.ac.uk. This author was supported in part by the EPSRC grant GR/R46641.

‡Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL, 60208-3118, USA. These authors were supported by National Science Foundation grants CCR-9987818, ATM-0086579 and CCR-0219438 and Department of Energy grant DE-FG02-87ER25047-A004.

1 Introduction

Sequential quadratic programming (SQP) constitutes one of the most successful methods for large-scale nonlinear optimization. In recent years, active set SQP methods have proven to be quite effective at solving problems with thousands of variables and constraints [9, 11], but are likely to become very expensive as the problems they are asked to solve become larger and larger. This is particularly so since the principal cost of SQP methods is often dominated by that of solving the quadratic programming (QP) subproblems, and in the absence of a QP truncating scheme, this cost can become prohibitive. These concerns have motivated us to look for a different active-set approach.

In this paper we describe a trust-region algorithm for nonlinear programming that does not require the solution of a general quadratic program at each iteration. It can be viewed as a so-called "EQP form" [12] of sequential quadratic programming, in which a guess of the active set is made (using linear programming techniques) and then an equality constrained quadratic program is solved to attempt to achieve optimality.

The idea of solving a linear program (LP) to identify an active set, followed by the solution of an equality constrained quadratic problem (EQP) was first proposed and analyzed by Fletcher and Sainz de la Maza [10], and more recently by Chin and Fletcher [4], but has received little attention beyond this. This "sequential linear programming-EQP method", or SLP-EQP in short, is motivated by the fact that solving quadratic subproblems with inequality constraints, as in the SQP method, can be prohibitively expensive for many large problems. The cost of solving one linear program followed by an equality constrained quadratic problem, could be much lower.

In this paper we go beyond the ideas proposed by Fletcher and Sainz de la Maza in that we investigate new techniques for generating the step, managing the penalty parameter and updating the LP trust region. Our algorithm also differs from the approach of Chin and Fletcher, who use a filter to determine the acceptability of the step, whereas we employ an ℓ_1 merit function. All of this results in major algorithmic differences between our approach and those proposed in the literature.

2 Overview of the Algorithm

The nonlinear programming problem will be written as

$$\min_{x} \quad f(x) \tag{2.1a}$$

$$\text{subject to} \quad h_i(x) = 0, \quad i \in \mathcal{E} \tag{2.1b}$$

$$g_i(x) \geq 0, \quad i \in \mathcal{I}, \tag{2.1c}$$

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 SLP-EQP algorithm studied in this paper is a trust-region method which uses a merit function to determine the acceptability of a step. It separates the active-set identification phase from the step computation phase — unlike SQP methods where both tasks are accomplished by solving a quadratic program — and employs different trust regions for each phase. First, a linear programming problem (LP) based on a linear model of the merit function is solved. The solution of this LP defines a step, d^{LP}, and a working set $\mathcal{W}$ which is a subset of the constraints active at the solution of this LP. Next a Cauchy step, d^{C}, is computed by minimizing a quadratic model of the merit function along the direction d^{LP}. The Cauchy step plays a crucial role in the global convergence properties of the algorithm. Once the LP and Cauchy steps have been computed, an equality constrained quadratic program (EQP) is solved, treating the constraints in $\mathcal{W}$ as equality constraints and ignoring all other constraints, to obtain the EQP point x^{EQP}.

The trial point x^{T} of the algorithm is chosen to lie on the line segment starting at the Cauchy point $x^{\mathrm{C}} = x + d^{\mathrm{C}}$ and terminating at the EQP point x^{EQP}, where x denotes the current iterate. The trial point x^{T} is accepted if it provides sufficient decrease of the merit function; otherwise the step is rejected, the trust region is reduced and a new trial point is computed.

The algorithm is summarized below. Here $\phi(x;\nu)$ denotes the ℓ_1 merit function

$$\phi(x;\nu) = f(x) + \nu \sum_{i\in\mathcal{E}} |h_i(x)| + \nu \sum_{i\in\mathcal{I}} (\max(0, -g_i(x)), \tag{2.2}$$

with penalty parameter ν. A quadratic model of ϕ will be denoted by q. The trust-region radius for the LP subproblem is denoted by Δ^{LP}, whereas Δ is the primary (master) trust-region radius that controls both the size of the EQP step and the total step.

Algorithm 2.1: SLP-EQP – General Outline

Initial data: x_0, $\Delta_0 > 0$, $\Delta_0^{\text{LP}} > 0$, $0 < \rho_u \le \rho_s < 1$.

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

1. LP point Solve an LP with trust region Δ_k^{LP} based on linear models of (2.1) to obtain step d_k^{LP}, and working set $\mathcal{W}_k$.

2. Cauchy point Compute $\alpha_k^{\text{LP}} \in (0,1]$ as the (approximate) minimizer of $q(\alpha d_k^{\text{LP}})$ such that $\alpha_k^{\text{LP}} \|d_k^{\text{LP}}\| \le \Delta_k$. Set $d_k^{\text{C}} = \alpha_k^{\text{LP}} d_k^{\text{LP}}$ and $x_k^{\text{C}} = x_k + d_k^{\text{C}}$.

3. EQP point Compute x^{EQP} by solving an EQP with trust region Δ_k and constraints defined by $\mathcal{W}_k$. Define $d_k^{\text{CE}} = x_k^{\text{EQP}} - x_k^{\text{C}}$ as the segment leading from the Cauchy point to the EQP point.

4. Trial point Compute $\alpha_k^{\text{EQP}} \in [0,1]$ as the (approximate) minimizer of $q(d_k^{\text{C}} + \alpha d_k^{\text{CE}})$. Set $d_k = d_k^{\text{C}} + \alpha_k^{\text{EQP}} d_k^{\text{CE}}$ and $x_k^{\text{T}} = x_k + d_k$.

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

6a. If $\rho_k \ge \rho_s$, choose
$$\Delta_{k+1} \ge \Delta_k,$$
otherwise choose
$$\Delta_{k+1} < \Delta_k.$$

6b. If $\rho_k \ge \rho_u$, set
$$x_{k+1} \leftarrow x_k^{\text{T}},$$
otherwise set
$$x_{k+1} \leftarrow x_k.$$

7. Compute Δ_{k+1}^{LP}.

An appealing feature of the SLP-EQP algorithm is that established techniques for solving large-scale versions of the LP and EQP subproblems are readily available. Current high quality LP software is capable of solving problems with more than a million variables and constraints, and the solution of an EQP can be performed efficiently using an iterative approach such as the conjugate gradient method. Two of the key questions regarding the SLP-EQP approach which will play a large role in determining its efficiency are: (i) how well does the linear program predict the optimal active set, and (ii) what is the cost of the iteration compared to its main competitors, the interior point and active-set approaches?

Many details of the algorithm are yet to be specified. This will be the subject of the following sections.

3 The Linear Programming (LP) Phase

The goal of the LP phase is to make an estimate of the optimal active set $\mathcal{W}^*$, at moderate cost. In general terms we want to solve the problem

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \nabla f(x)^T d && \text{(3.1a)} \\ \text{subject to} \quad & h_i(x) + \nabla h_i(x)^T d = 0, \quad i \in \mathcal{E} && \text{(3.1b)} \\ & g_i(x) + \nabla g_i(x)^T d \geq 0, \quad i \in \mathcal{I} && \text{(3.1c)} \\ & \|d\|_\infty \leq \Delta^{\text{LP}}, && \text{(3.1d)} \end{aligned}$$

where Δ^{LP} is a trust-region radius whose choice will be discussed in Section 3.1. The working set $\mathcal{W}$ is defined as the set of constraints that are active at the solution of this LP, if these constraints are linearly independent. Otherwise, some linearly independent subset of these form the working set.

Working with this LP is attractive since it requires no choice of parameters, but it has the drawback that its constraints may be infeasible. This possible inconsistency of constraint linearizations and the trust region has received considerable attention in the context of SQP methods; see, e.g. [5] and the references therein.

To deal with the possible inconsistency of the constraints we follow an ℓ_1-penalty approach in which the constraints (3.1*b*)–(3.1*c*) are incorporated in the form of a penalty term in the model objective. Specifically, we reformulate the LP phase as the minimization of a linear approximation of the ℓ_1 merit function (2.2) subject to the trust-region constraint. The linear approximation of the merit function ϕ at the current estimate x is given by

$$\begin{aligned} \ell(d) \quad = \quad & \nabla f(x)^T d + \nu \sum_{i \in \mathcal{E}} |h_i(x) + \nabla h_i(x)^T d| \\ & + \nu \sum_{i \in \mathcal{I}} \max(0, -g_i(x) - \nabla g_i(x)^T d). \end{aligned}$$

The working-set determination problem is then given by

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \ell(d) \\ \text{subject to} \quad & \|d\|_\infty \leq \Delta^{\text{LP}}. \end{aligned}$$

The function ℓ is non-differentiable but it is well-known that this problem can be written as the following equivalent, smooth linear program

$$\begin{aligned} \underset{d,r,s,t}{\text{minimize}} \quad & \nabla f(x)^T d + \nu \sum_{i \in \mathcal{E}} (r_i + s_i) + \nu \sum_{i \in \mathcal{I}} t_i && \text{(3.4a)} \\ \text{subject to} \quad & h_i(x) + \nabla h_i(x)^T d = r_i - s_i, \quad i \in \mathcal{E} && \text{(3.4b)} \\ & g_i(x) + \nabla g_i(x)^T d \geq -t_i, \quad i \in \mathcal{I} && \text{(3.4c)} \\ & \|d\|_\infty \leq \Delta^{\text{LP}} && \text{(3.4d)} \\ & r, s, t \geq 0. && \text{(3.4e)} \end{aligned}$$

Here r, s and t are vectors of slack variables which allow for the relaxation of the equality and inequality constraints. We denote a solution of this problem by $d^{\text{LP}}(\nu)$.

The working set $\mathcal{W}$ is defined as some linearly independent subset of the active set $\mathcal{A}$ at the LP solution point which is defined as

$$\begin{aligned}\mathcal{A}(d^{\text{LP}}) = \quad & \{i \in \mathcal{E} \mid h_i(x) + \nabla h_i(x)^T d^{\text{LP}} = 0\} \cup \\ & \{i \in \mathcal{I} \mid g_i(x) + \nabla g_i(x)^T d^{\text{LP}} = 0\}.\end{aligned}$$

Software for linear programming typically provides this linearly independent set. If the LP subproblem is non-degenerate the working set is synonymous with the active set defined above. Note that we do not include all of the equality constraints in the active set but only those whose right hand side is zero in (3.4b), for otherwise the EQP system could be overly constrained.

In our software implementation, simple bound constraints on the variables are omitted from the merit function and handled as explicit constraints. We will ensure that the starting point and all subsequent iterates satisfy the bounds. In particular we add lower and upper bounds to (3.4) to ensure that the LP step satisfies the bounds. For the sake of simplicity, however, we will omit all details concerning the handling of bounds constraints, and will only make remarks about them when pertinent.

3.1 Trust Region for the LP Step

Since the model objective (3.1a) is linear, the choice of the trust-region radius Δ^{LP} is much more delicate than in trust-region methods that employ quadratic models. The trust region must be large enough to allow significant progress toward the solution, but must be small enough so that the LP subproblem identifies only locally active constraints. We have found that it is difficult to balance these two goals, and will present here a strategy that appears to work well in practice and is supported by a global convergence theory. There may, however, be more effective strategies and the choice of Δ^{LP} remains an open subject of investigation.

We update the LP trust region as follows. If the trial step d_k taken by the algorithm on the most current iteration was accepted (i.e., $\rho_k \geq \rho_u$), we define

$$\Delta_{k+1}^{\text{LP}} = \begin{cases} \min(\max\{1.2\|d_k\|_\infty, 1.2\|d_k^{\text{C}}\|_\infty, 0.1\Delta_k^{\text{LP}}\}, 7\Delta_k^{\text{LP}}), & \text{if} \quad \alpha_k^{\text{LP}} = 1 \\ \min(\max\{1.2\|d_k\|_\infty, 1.2\|d_k^{\text{C}}\|_\infty, 0.1\Delta_k^{\text{LP}}\}, \Delta_k^{\text{LP}}), & \text{if} \quad \alpha_k^{\text{LP}} < 1 \end{cases}, \tag{3.5}$$

whereas if the step d_k was rejected we set

$$\Delta_{k+1}^{\text{LP}} = \min(\max\{0.5\|d_k\|_\infty, 0.1\Delta_k^{\text{LP}}\}, \Delta_k^{\text{LP}}). \tag{3.6}$$

The motivation for (3.5) stems from the desire that Δ_{k+1}^{LP} be no larger than a multiple of the norm of the trial step d_k and the Cauchy step d_k^{C}, so that the LP trust region be small enough to exclude extraneous, inactive constraints as the iterate converges to a solution. Note that the LP trust region can decrease after an accepted step, and we include the term $0.1\Delta_k^{\text{LP}}$ to limit the rate of this decrease. Finally, we only allow the LP trust region to increase if the step is accepted and $\alpha_k^{\text{LP}} = 1$ (i.e., $d_k^{\text{LP}} = d_k^{\text{C}}$). Otherwise, if $\alpha_k^{\text{LP}} < 1$ this may be an indication that out linear models are not accurate. The term $7\Delta_k^{\text{LP}}$ prevents the LP trust region from growing too rapidly in the case when it may increase.

When the trial step d_k is rejected, (3.6) ensures that Δ_k^{LP} does not grow. We would again want to make Δ_{k+1}^{LP} a fraction of $\|d_k\|_\infty$, and the term $0.1\Delta_k^{\text{LP}}$ limits the rate of decrease.

This LP trust-region update is supported by the global convergence theory presented in Byrd et al [2], which also provides a range of admissible values for the constants in (3.5)–(3.6). The results in [2] show in particular that, if the sequence of iterates $\{x_k\}$ is bounded, then there exists an accumulation point of the SLP-EQP algorithm that is stationary for the merit function ϕ.

4 The Cauchy Point

The reduction in the objective and constraints provided by the LP step can be very small. To ensure that the algorithm has favorable global convergence properties, we will require that the total step makes at least as much progress as a *Cauchy point* x^{C}. This is a point which provides sufficient decrease of a quadratic model of the merit function along the LP direction d^{LP} and subject to the restriction $\|x^{\text{C}} - x\|_2 \le \Delta$. The quadratic model, $q(d)$, is defined as

$$q(d) = \ell(d) + \tfrac{1}{2} d^T H(x,\lambda) d, \tag{4.1}$$

where $H(x,\lambda)$ denotes the Hessian of the Lagrangian of the NLP problem (2.1) and λ is a vector of Lagrange multiplier estimates defined later on in (5.7). To define the Cauchy point, we select $0 < \tau < 1$, let $\delta = \min(1, \Delta/||d^{\text{LP}}\|_2)$ and compute a steplength $0 < \alpha^{\text{LP}} \le 1$ as the first member of the sequence $\{\delta\tau^i\}_{i=0,1,\ldots}$ for which

$$\phi(x;\nu) - q(\alpha^{\text{LP}} d^{\text{LP}}) \ge \eta[\phi(x;\nu) - \ell(\alpha^{\text{LP}} d^{\text{LP}})], \tag{4.2}$$

where $\eta \in (0,1)$ is a given constant. We then define

$$x^{\text{C}} = x + \alpha^{\text{LP}} d^{\text{LP}} \equiv x + d^{\text{C}}. \tag{4.3}$$

The backtracking line search used to compute α^{LP} does not involve evaluations of the problem functions, but rather, only evaluations of their inexpensive model approximations.

5 The EQP Step

Having computed the LP step d^{LP} which determines the working set $\mathcal{W}$, we now wish to compute a step d that attempts to achieve optimality for this working set by solving an equality constrained quadratic program (EQP) of the form

$$\begin{aligned}
\underset{d}{\text{minimize}} \quad & \tfrac{1}{2} d^T H(x,\lambda) d + \nabla f(x)^T d && \text{(5.1a)}\\
\text{subject to} \quad & h_i(x) + \nabla h_i(x)^T d = 0, \quad i \in \mathcal{E} \cap \mathcal{W} && \text{(5.1b)}\\
& g_i(x) + \nabla g_i(x)^T d = 0, \quad i \in \mathcal{I} \cap \mathcal{W} && \text{(5.1c)}\\
& \|d\|_2 \le \Delta. && \text{(5.1d)}
\end{aligned}$$

The trust-region radius Δ places some restriction on the step size and prevents the step from being unbounded in the case of negative curvature. Note that the constraints (5.1b)–(5.1c) are consistent by definition of the working set $\mathcal{W}$, but to make them compatible with the trust region we may relax them, as will be explained below.

Let $A_{\mathcal{W}} \in I\!R^{p\times n}$ represent the Jacobian matrix of the constraints in the working set where p is the number of constraints in the working set, and define a matrix $Z_{\mathcal{W}} \in I\!R^{n\times(n-p)}$ which is a null-space basis for $A_{\mathcal{W}}$ (i.e., $A_{\mathcal{W}} Z_{\mathcal{W}} = 0$). One can express the solution of (5.1) as

$$d = d^{\mathrm{N}} + Z_{\mathcal{W}} d^{\mathrm{Z}}, \tag{5.2}$$

for some vector d^{N} which satisfies the constraints (5.1b)–(5.1c) and some reduced space vector $d^{\mathrm{Z}} \in I\!R^{n-p}$. The vector d^{N} will be computed here as

$$d^{\mathrm{N}} = \alpha^{\mathrm{N}} \hat{d}^{\mathrm{N}}, \tag{5.3}$$

where $\hat{d}^{\mathrm{N}}$ is the minimum norm solution of (5.1b)–(5.1c) and α^{N} is the largest value in $[0,1]$ for which $\|\alpha \hat{d}^{\mathrm{N}}\|_2 \le 0.8\Delta$. (If $\alpha^{\mathrm{N}} < 1$, we replace the zeros in the right hand sides of (5.1b) and (5.1c) by

$$r_{\mathcal{E}} = h_i(x) + \nabla h_i(x)^T d^{\mathrm{N}},\ i \in \mathcal{E} \cap \mathcal{W}, \quad r_{\mathcal{I}} = g_i(x) + \nabla g_i(x)^T d^{\mathrm{N}},\ i \in \mathcal{I} \cap \mathcal{W}.)$$

If we define $\bar{d} = Z_{\mathcal{W}} d^{\mathrm{Z}}$ as a step in the null-space of the working set constraint gradients, then we can compute the EQP step d^{EQP} as an approximate solution of the problem

$$\begin{aligned} \underset{\bar{d}}{\text{minimize}} \quad & \tfrac{1}{2}\bar{d}^T H^{\mathrm{EQP}}(x,\lambda)\bar{d} + (g^{\mathrm{EQP}})^T \bar{d} && \text{(5.4a)} \\ \text{subject to} \quad & \nabla h_i(x)^T \bar{d} = 0, \quad i \in \mathcal{E} \cap \mathcal{W} && \text{(5.4b)} \\ & \nabla g_i(x)^T \bar{d} = 0, \quad i \in \mathcal{I} \cap \mathcal{W} && \text{(5.4c)} \\ & \|\bar{d}\|_2 \le \Delta^{\mathrm{EQP}}, && \text{(5.4d)} \end{aligned}$$

where the definitions of the matrix $H^{\mathrm{EQP}}(x,\lambda)$ and the vector g^{EQP} are discussed below, and

$$\Delta^{\mathrm{EQP}} = \sqrt{\Delta^2 - \|d^{\mathrm{N}}\|_2^2}.$$

The EQP point is computed as

$$x^{\mathrm{EQP}} = x + d^{\mathrm{N}} + d^{\mathrm{EQP}}. \tag{5.5}$$

The matrix H^{EQP} should approximate the Hessian of the Lagrangian of the NLP problem (2.1), and this raises the question of how to compute the Lagrange multipliers that define H^{EQP}. As discussed in section 8, we will generally use a least squares multipliers approach in which inactive constraints are assigned zero multipliers. This choice is appropriate for the termination test and most other aspects of the algorithm, but is inadequate for the EQP step computation. The reason is that by setting the multipliers for the inactive constraints to zero we ignore curvature information concerning violated constraints — and this can lead to inefficiencies, as we have observed in practice. Instead of this, for the EQP step we will obtain Lagrange multiplier estimates for the inactive constraints based on our ℓ_1 penalty function.

Let us define the set of violated general constraints for the projection step d^{N} as

$$\mathcal{V} = \{i \notin \mathcal{W} \mid h_i(x) + \nabla h_i(x)^T d^{\mathrm{N}} \ne 0\} \cup \{i \notin \mathcal{W} \mid g_i(x) + \nabla g_i(x)^T d^{\mathrm{N}} < 0\}, \tag{5.6}$$

and denote its complement by $\mathcal{V}^c$. The Hessian of the quadratic model (5.4a) will be defined as

$$\begin{aligned} H^{\text{EQP}}(x,\lambda) = \quad & \nabla^2 f(x) + \nu \sum_{i\in\mathcal{V}\cap\mathcal{E}} \text{sign}(h_i(x) + \nabla h_i(x)^T d^{\text{N}})\nabla^2 h_i(x) \\ & -\nu \sum_{i\in\mathcal{V}\cap\mathcal{I}} \nabla^2 g_i(x) - \sum_{i\in\mathcal{V}^c\cap\mathcal{E}} \lambda_i \nabla^2 h_i(x) - \sum_{i\in\mathcal{V}^c\cap\mathcal{I}} \lambda_i \nabla^2 g_i(x). \end{aligned} \tag{5.7}$$

The terms involving ν in (5.7) are the Hessians of the penalty terms in the ℓ_1 function ϕ for the violated constraint indices. Since these penalty terms are inactive for the projection step d^{N}, they are smooth functions within some neighborhood of this point. The signs for these terms are based on the values of the linearization of these constraints at the projection point. We view (5.7) as the Hessian of the penalty function ϕ, where inactive, violated constraints have been assigned non-zero multipliers.

We can also incorporate linear information on the violated constraints into the EQP step by defining

$$\begin{aligned} g^{\text{EQP}} = \quad & H^{\text{EQP}}(x,\lambda)d^{\text{N}} + \nabla f(x) \\ & +\nu \sum_{i\in\mathcal{V}\cap\mathcal{E}} \text{sign}(h_i(x) + \nabla h_i(x)^T d^{\text{N}})\nabla h_i(x) - \nu \sum_{i\in\mathcal{V}\cap\mathcal{I}} \nabla g_i(x). \end{aligned} \tag{5.8}$$

The last three terms in (5.8) represent the gradient of the terms in the penalty function whose linearization is nonconstant on the working set subspace.

To summarize, these definitions are necessitated by the active-set approach followed in this paper. In a classical SQP method, the QP solver typically enforces that the linearized constraints are satisfied throughout the step computation process. In this case, it is not necessary to include curvature information on violated constraints since the violated set $\mathcal{V}$ would be empty. By contrast our algorithm may completely ignore some of the constraints in the EQP phase and we need to account for this.

5.1 Solution of the EQP

The equality constrained quadratic problem (5.4), with its additional spherical trust-region constraint, will be solved using a projected Conjugate-Gradient/Lanczos iteration, as implemented in the GALAHAD code `GLTR` of Gould et al [14] (HSL routine `VF05` [15]). This algorithm has the feature of continuing for a few more iterations after the first negative curvature direction is encountered.

The projected CG/Lanczos approach applies orthogonal projections at each iteration to keep $\bar{d}$ in the null-space of $A_{\mathcal{W}}$. The projection of a vector v, say $w = Pv$, is computed by solving the system

$$\begin{bmatrix} I & A_{\mathcal{W}}^T(x) \\ A_{\mathcal{W}}(x) & 0 \end{bmatrix} \begin{bmatrix} w \\ u \end{bmatrix} = \begin{bmatrix} v \\ 0 \end{bmatrix} \tag{5.9}$$

where u is an auxiliary vector; see also [13]. We use the routine `MA27` from the HSL library [15] to factor this system.

The CG iteration can be preconditioned to speed up convergence by replacing the identity matrix in the (1,1) block of the coefficient matrix in (5.9) with a preconditioner

G which in some sense approximates H^{EQP}. However, we will not consider preconditioners here since they require significant changes to various aspects of our algorithm.

6 The Trial Step

Having computed the LP, Cauchy and EQP steps, we now combine them to define the trial step of the iteration, d, in such a way as to obtain sufficient decrease in the quadratic model of the penalty function.

We consider the vector leading from the Cauchy point to the EQP point,

$$d^{\text{CE}} = x^{\text{EQP}} - x^{\text{C}},$$

where x^{C} and x^{EQP} are defined in (4.3) and (5.5), respectively. We then compute the steplength $\alpha^{\text{EQP}} \in [0,1]$ which approximately minimizes $q(d^{\text{C}} + \alpha d^{\text{CE}})$, where q is given by (4.1). (If some bounds of the NLP are violated, we decrease α^{EQP} further so that they are satisfied.) The trial step of the iteration will be defined as

$$d = d^{\text{C}} + \alpha^{\text{EQP}} d^{\text{CE}},$$

where d^{C} is the step to the Cauchy point. In practice we do not implement an exact line search to compute α^{EQP}, but rather use a backtracking line search that terminates as soon as the model q has decreased.

The computation of the trial step d is similar to the dogleg method of Powell [19, 20] for approximately minimizing a quadratic objective subject to a trust-region constraint. As in the dogleg approach, the step is computed via a one dimensional line search along a piecewise path from the origin to the Cauchy point x^{C} to a Newton-like point (the EQP point x^{EQP}). However, in contrast to the standard dogleg method, the model q is not necessarily a decreasing function along the segment from the Cauchy point to the EQP point when the Hessian is positive-definite (which is why a line search is used to compute α^{EQP}). Since the minimizer can occur at x^{C} we set $\alpha^{\text{EQP}} = 0$ if it becomes very small (in our tests, less than 10^{-16}).

7 Step Acceptance, Trust Region Update and SOC

Given a current point x and penalty parameter ν, a trial point, x^{T} given by a step d is accepted if

$$\rho = \frac{\text{ared}}{\text{pred}} = \frac{\phi(x;\nu) - \phi(x^{\text{T}};\nu)}{q(0) - q(d)} > \rho_u, \quad \rho_u \in [0,1). \tag{7.1}$$

In our implementation we set $\rho_u = 10^{-8}$. Since we always ensure that the predicted reduction is positive (by the choices of α^{LP} and α^{EQP} used to compute the trial step d), the acceptance rule (7.1) guarantees that we only accept steps which give a reduction in the merit function.

As is well known (Maratos [18]) steps that make good progress toward the solution may be rejected by the penalty function ϕ, which may lead to slow convergence. We address this difficulty by computing a second order correction (SOC) step [8], which incorporates second order curvature information on the constraints.

If the trial point x^{T} does not provide sufficient decrease of the merit function, we compute d^{SOC} as the minimum norm solution of

$$A_{\mathcal{W}}(x)d + c_{\mathcal{W}}(x^{\mathrm{T}}) = 0,$$

where $c_{\mathcal{W}}(x^{\mathrm{T}})$ is the value of the constraints in the working set at the original trial point. In this case the trial step is computed as the sum of the original trial step and some fraction of the second order correction step, d^{SOC}

$$d \leftarrow d + \tau^{\mathrm{SOC}} d^{\mathrm{SOC}},$$

where, the scalar $\tau^{\mathrm{SOC}} \in [0,1]$ enforces satisfaction of all of the bounds on the variables.

In our algorithm we compute d^{SOC} by solving the linear system

$$\begin{bmatrix} I & A_{\mathcal{W}}^T(x) \\ A_{\mathcal{W}}(x) & 0 \end{bmatrix} \begin{bmatrix} d^{\mathrm{SOC}} \\ t \end{bmatrix} = \begin{bmatrix} 0 \\ -c_{\mathcal{W}}(x^{\mathrm{T}}) \end{bmatrix}. \tag{7.2}$$

Note that the computation of the second order correction step takes into account only the constraints in the current working set (ignoring other constraints). The motivation for this is twofold. First, it allows us to use the same coefficient matrix in (7.2) as is used to compute projections in the CG/Lanczos routine of the EQP step (5.9) and therefore no additional matrix factorizations are needed. Second, in the case when our working set is accurate, we are justified in ignoring the constraints not in the working set in the SOC step computation. Conversely, if our working set is very inaccurate it is unlikely that a SOC step that would include all the constraints would be of much value anyway.

The SOC step could be computed selectively but for simplicity we take the conservative approach of attempting a SOC step after every rejected trial step. Another issue to consider is from where to attempt the SOC step. There appear to be two viable options, the trial point, $x^{\mathrm{T}} = x + d$, and the EQP point x^{EQP}. If we attempt the SOC step from the full EQP point, this requires an extra evaluation of the objective and constraint functions (assuming $x^{\mathrm{T}} \neq x^{\mathrm{EQP}}$). For this reason we attempt the SOC step from the original trial point.

For a current trust-region radius Δ_k and step d_k, we update the (master) trust-region radius by the following rule

$$\Delta_{k+1} = \begin{cases} \max(\Delta_k, 7\|d_k\|_2), & \text{if} \quad 0.9 \le \rho \\ \max(\Delta_k, 2\|d_k\|_2), & \text{if} \quad 0.3 \le \rho < 0.9 \\ \Delta_k, & \text{if} \quad 10^{-8} \le \rho < 0.3 \\ \min(0.5\Delta_k, 0.5\|d_k\|_2), & \text{if} \quad \rho < 10^{-8} \end{cases}, \tag{7.3}$$

where ρ is defined in (7.1) and represents the agreement between the reduction in the merit function and the reduction predicted by the quadratic model q.

8 The Lagrange Multiplier Estimates

Both the LP and the EQP phases of the algorithm provide possible choices for Lagrange multiplier estimates. However, we choose to compute least-squares Lagrange multipliers since they satisfy the optimality conditions as well as possible for the given iterate x, and can be computed very cheaply as we now discuss.

The multipliers corresponding to the constraints in the current working set $\lambda_{\mathcal{W}}$ are computed by solving the system

$$\begin{bmatrix} I & A_{\mathcal{W}}^T(x) \\ A_{\mathcal{W}}(x) & 0 \end{bmatrix} \begin{bmatrix} t \\ \lambda_{\mathcal{W}} \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix}. \tag{8.1}$$

Since the coefficient matrix in the system above needs to be factored to compute projections (5.9) in the CG/Lanczos method, the cost of computing these least-squares multipliers is one extra backsolve which is a negligible cost in the overall iteration (considering the CG/Lanczos method involves n_{CG} backsolves where n_{CG} is the number of CG/Lanczos iterations performed during the EQP phase). If any of the computed least-squares multipliers corresponding to inequality constraints are negative, these multipliers are reset to zero. The Lagrange multipliers λ corresponding to constraints not in the current working set are set to zero (except in the computation of the Hessian of the Lagrangian $H(x, \lambda)$ where they are assigned a penalty-based value as indicated by (5.7)). These least squares multipliers are used in the stopping test for the nonlinear program.

9 Penalty Parameter Update

The choice of the penalty parameter ν in (2.2) has a significant impact on the performance of the iteration. If the algorithm is struggling to become feasible, it can be beneficial to increase ν. However, if ν becomes too large too quickly this can cause the algorithm to converge very slowly. Existing strategies for updating the penalty parameter are based on tracking the size of the Lagrange multipliers or checking the optimality conditions for the non-differentiable merit function ϕ.

Here we propose a new approach for updating the penalty parameter based on the LP phase. We take the view that, if it is possible to satisfy the constraints (3.1b)–(3.1d), then we would like to choose ν large enough in (3.4), to do so. Otherwise, if this is not possible, then we choose ν to enforce a sufficient decrease in the violation of the linearized constraints at x, which we measure through the function

$$\xi(x, \nu) = \frac{1}{|\mathcal{E}| + |\mathcal{I}|} \left[\sum_{i \in \mathcal{E}} |h_i(x) + \nabla h_i(x)^T d^{\text{LP}}(\nu)| + \sum_{i \in \mathcal{I}} \max(0, -g_i(x) - \nabla g_i(x)^T d^{\text{LP}}(\nu)) \right].$$

The minimum possible infeasibility value for the LP subproblem will be denoted by $\xi(x, \nu_\infty)$, where ν_∞ is meant to represent an infinite value for the penalty parameter.

At the current point x_k and given ν_{k-1} from the previous iteration, we use the following relation to define the sufficient decrease in infeasibility required by the new penalty parameter ν_k:

$$\xi(x_k, \nu_{k-1}) - \xi(x_k, \nu_k) \geq \epsilon(\xi(x_k, \nu_{k-1}) - \xi(x_k, \nu_\infty)), \quad \epsilon \in (0, 1]. \tag{9.1}$$

In our implementation we use the value $\epsilon = 0.1$. We can now outline our strategy for updating the penalty parameter on each iteration.

Algorithm 9.1 *Penalty Parameter Update Strategy*

Given: $(x_k,\ \nu_{k-1})$ *and the parameters* ν_∞, tol_1, tol_2 *and* ϵ.
Solve LP (3.4) *with* $(x_k,\ \nu_{k-1})$ *to get* $d^{LP}(\nu_{k-1})$.
if $d^{LP}(\nu_{k-1})$ *is feasible (i.e.,* $\xi(x_k,\nu_{k-1}) < tol_1$*)*
  $\nu_k \leftarrow \nu_{k-1}$ *(Case 1).*
else
  Solve LP (3.4) *with* $(x_k,\ \nu_\infty)$ *to get* $d^{LP}(\nu_\infty)$.
  if $d^{LP}(\nu_\infty)$ *is feasible (i.e.,* $\xi(x_k,\nu_\infty) < tol_1$*)*
    Choose some $\nu_{k-1} < \nu_k \le \nu_\infty$ *such that* $\xi(x_k,\nu_k) < tol_1$ *(Case 2).*
  else if $\xi(x_k,\nu_{k-1}) - \xi(x_k,\nu_\infty) < tol_2$ *(no significant progress in feasibility possible)*
    $\nu_k \leftarrow \nu_{k-1}$ *(Case 3).*
  else
    Choose some $\nu_{k-1} < \nu_k \le \nu_\infty$ *such that* (9.1) *is satisfied (Case 4).*
  end (if)
end (if)

In our implementation we set $tol_1 = tol_2 = 10^{-8}$, and initialize $\nu_{\text{init}} = 10$ where ν_{init} is the value of the penalty parameter at the beginning of the algorithm. In practice, instead of using a very large penalty value for computing $\xi(x_k,\nu_\infty)$, this value is computed by setting $\nabla f = 0$ in the linear objective (3.4a) which has the effect of ignoring the NLP objective $f(x_k)$ and minimizing the linear constraint violation as much as possible.

The implementation of Case 2 is achieved by increasing ν_{k-1} by a factor of ten and re-solving the LP until feasibility is achieved. Case 4 is implemented in a similar manner until the condition (9.1) is satisfied with $\epsilon = 0.1$. In Case 3 we determine that no significant improvement in feasibility is possible for the current LP (as determined by comparing the feasibility measure for ν_{k-1} with the feasibility measure for ν_∞) and so we set $\nu_k \leftarrow \nu_{k-1}$ rather than increasing the penalty parameter.

One concern with our penalty parameter update strategy is that it may require the solution of multiple LPs per iteration. However, in practice this is only the case generally in a small fraction of the total iterations. Typically the penalty parameter only increases early on in the optimization calculation and then settles down to an acceptable value for which the algorithm achieves feasibility. Moreover, it is our experience that although this may result in multiple LP solves on some iterations, it results in an overall savings in iterations (and total LP solves) by achieving a better penalty parameter value more quickly, compared with rules which update the penalty parameter based on the size of the Lagrange multipliers. In addition, we have observed that, when using a simplex LP solver, the extra LP solves are typically very inexpensive requiring relatively few simplex iterations because of the effectiveness of warm starts when re-solving the LP with a different penalty parameter value. (In the results reported in Section 11 the percentage of additional simplex iterations required by Algorithm 9.1 averages roughly 4%.)

Another concern is that using this scheme the penalty parameter may become too large too quickly and we may need to add a safeguard which detects this and reduces ν on occasion. In practice we have noticed that this does seem to occur on a small minority of the problems and we have implemented the following strategy for reducing ν. If there is a sequence of five consecutive successful iterations where the iterate is feasible and $\nu > 1000(\|\lambda\|_\infty + 1)$, then ν is judged to be too large and is reset to $\nu = \|\lambda\|_\infty + 10$. The penalty parameter ν is permitted to be decreased a maximum of two times. Although this approach is somewhat conservative, it has proved to be quite successful in practice in handling the few problems where ν becomes too large without adversely affecting the majority of problems where it does not.

10 The Complete Algorithm

We now summarize the algorithm using the pseudo-code below. We will call our particular implementation of the SLP-EQP method the SLIQUE Algorithm.

Algorithm 10.1: SLIQUE **Algorithm**

Initial data: x_0, $\Delta_0 > 0$, $\Delta_0^{\text{LP}} > 0$, $0 < \rho_u < 1$.

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

1. LP point Use Algorithm 9.1 to obtain step d_k^{LP} and penalty parameter ν_k.

2. Working set Define the working set $\mathcal{W}_k$ at $x_k + d_k^{\text{LP}}$, as described in section 3.

3. Multipliers Compute least squares multipliers $\lambda_{\mathcal{W}}$ by solving (8.1).
Set $\lambda_i = [\lambda_{\mathcal{W}}]_i$, $i \in \mathcal{W}$; $\lambda_i = 0$, $i \notin \mathcal{W}$. If $\lambda_i < 0$ for $i \in \mathcal{I}$, set $\lambda_i = 0$.

4. Evaluate Hessian (5.7), at x_k.

5. Cauchy point Compute $\alpha_k^{\text{LP}} \in (0, 1]$ as the (approximate) minimizer of $q(\alpha d_k^{\text{LP}})$ such that $\alpha_k^{\text{LP}} \| d_k^{\text{LP}} \| \leq \Delta_k$. Set $d_k^{\text{C}} = \alpha_k^{\text{LP}} d_k^{\text{LP}}$ and $x_k^{\text{C}} = x_k + d_k^{\text{C}}$.

6. EQP point Compute x^{EQP} from (5.5) using (5.3) and the solution of (5.4), with constraints defined by $\mathcal{W}_k$. Define $d_k^{\text{CE}} = x_k^{\text{EQP}} - x_k^{\text{C}}$ as the segment leading from the Cauchy point to the EQP point.

7. Trial point Compute $\alpha_k^{\text{EQP}} \in [0, 1]$ as the (approximate) minimizer of $q(d_k^{\text{C}} + \alpha d_k^{\text{CE}})$ (and such that all the simple bounds of the problem are satisfied). Set $d_k = d_k^{\text{C}} + \alpha_k^{\text{EQP}} d_k^{\text{CE}}$, $x_k^{\text{T}} = x_k + d_k$, `trySOC = True`.

8. Evaluate $f(x_k^{\text{T}})$, $h(x_k^{\text{T}})$ and $g(x_k^{\text{T}})$.

9. Compute

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

10. If $\rho_k \geq \rho_u$, set $x_{k+1} \leftarrow x_k^{\text{T}}$, $f(x_{k+1}) \leftarrow f(x_k^{\text{T}})$, $h(x_{k+1}) \leftarrow h(x_k^{\text{T}})$ and $g(x_{k+1}) \leftarrow g(x_k^{\text{T}})$. Evaluate $\nabla f(x_{k+1})$; $\nabla h_i(x_{k+1})$, $i \in \mathcal{E}$; and $\nabla g_i(x_{k+1})$, $i \in \mathcal{I}$. Go to Step 13.

11. If `trySOC = True`, compute d^{SOC} by solving (7.2) and choose τ^{SOC} so that the simple bounds of the problem are satisfied. Set $x_k^{\text{T}} = x_k + d_k + \tau^{\text{SOC}} d^{\text{SOC}}$, `trySOC = False`. Go to step 8.

12. Set $x_{k+1} \leftarrow x_k$.

13. Compute Δ_{k+1} by means of (7.3).

14. Compute Δ_{k+1}^{LP} using (3.5)–(3.6).

In our implementation the intial trust region radii are set to $\Delta_0 = 1$ and $\Delta_0^{\text{LP}} = 0.8\Delta_0/\sqrt{n}$, where n is the number of problem variables.

11 Numerical Tests

In order to assess the potential of the SLP-EQP approach taken in SLIQUE, we test it here on the CUTEr [1] set of problems and compare it with the state-of-the-art codes KNITRO [3, 22] and SNOPT [11].

SLIQUE 1.0 implements the algorithm outlined in the previous section. In all results reported in this section, SLIQUE 1.0 uses the commercial LP software package ILOG CPLEX 8.0 [16] running the default dual simplex approach to solve the LP subproblems. KNITRO 2.1 implements a primal-dual interior-point method with trust regions. It makes use of second derivative information, and controls the barrier parameter using a path-following approach. SNOPT 6.1-1(5) is a line search SQP method in which the search direction is determined by an active-set method for convex quadratic programming. SNOPT requires only first derivatives of the objective function and constraints, and maintains a (limited memory) BFGS approximation to the reduced Hessian of a Lagrangian function. Even though SNOPT uses only first derivatives (whereas KNITRO and SLIQUE use second derivatives) it provides a worthy benchmark for our purposes since it is generally regarded as one of the most effective active-set SQP codes available for large-scale nonlinear optimization.

All tests described in this paper were performed on a Sun Ultra 5, with 1Gb of memory running SunOS 5.7. All codes are written in Fortran, were compiled using the Sun f90 compiler with the "-O" compilation flag, and were run in double precision using all their default settings except as noted below. For SNOPT, the superbasics limit was increased to 2000 to allow for the solution of the majority of the CUTEr problems. However, for some problems this limit was still too small and so for these problems the superbasics limit was increased even more until it was sufficiently large. Limits of 1 hour of CPU time and 3000 outer or major iterations were imposed for each problem; if one of these limits was reached the code was considered to have failed. The stopping tolerance was set at 10^{-6} for all solvers. Although, it is nearly impossible to enforce a uniform stopping condition, the stopping conditions for SLIQUE and KNITRO were constructed to be very similar to that used in SNOPT.

The termination test used for SLIQUE in the numerical results reported in this section is as follows. Algorithm 10.1 stops when a primal-dual pair (x, λ) satisfies

$$\max\{||\nabla f(x) - A(x)^T\lambda||_\infty, ||g(x) * \lambda_g||_\infty\} \quad < \quad 10^{-6}(1 + ||\lambda||_2) \tag{11.1}$$

$$\text{and} \quad \max\left\{\max_{i\in\mathcal{E}}|h_i(x)|, \max_{i\in\mathcal{I}}(0, -g_i(x))\right\} \quad < \quad 10^{-6}(1 + ||x||_2), \tag{11.2}$$

where A denotes the Jacobian of all the constraints, and $g(x) * \lambda_g$ denotes the componentwise product of the vector g and its corresponding multipliers.

11.1 Robustness

In order to first get a picture of the robustness of the SLIQUE algorithm we summarize its performance on a subset of problems from the CUTEr test set (as of May 15, 2002). Since we are primarily interested in the performance of SLIQUE on general nonlinear optimization problems with inequality constraints and/or bounds on the variables (such that the active-set identification mechanism is relevant), we exclude all unconstrained problems and problems whose only constraints are equations or fixed variables. We also exclude LPs and feasibility problems (problems with zero degrees of freedom). In addition eight problems (`ALLINQP, CHARDIS0, CHARDIS1, CONT6-QQ, DEGENQP, HARKERP2, LUBRIF, ODNAMUR`) were removed because they could not be comfortably run within the memory limits of the testing machine for any of the codes. The remaining 560 problems form our test set. These remaining problems can be divided between three sets: quadratic programs (QP), problems whose only constraints are simple bounds on the variables (BC), and everything else, which we refer to as generally constrained (GC) problems. If a problem is a QP just involving bound constraints, it is included only in the BC set.

Although we will not show it here, the SLIQUE implementation of the SLP-EQP algorithm described in this paper is quite robust and efficient at solving simpler classes of problems (e.g., LPs, unconstrained problems, equality constrained problems and feasibility problems) as evidenced in [21].

We should note that there are a few problems in CUTEr for which a solution does not exist (for example the problem may be infeasible or unbounded). Although, it is important for a code to recognize and behave intelligently in these cases, we do not evaluate the ability of a code to do so here. For simplicity, we treat all instances where an optimal solution is not found as a failure regardless of whether or not it is possible to find such a point. Moreover, we do not distinguish between different local minima as it is often too difficult to do so, and because we have noted that in the vast majority of cases (roughly) the same point is found by the competing methods.

Problem class	Problem size	# of problems			
		QP	BC	GC	Total
VS	$1 \leq n+m < 100$	30	59	177	266
S	$100 \leq n+m < 1000$	24	5	47	76
M	$1000 \leq n+m < 10000$	27	30	61	118
L	$10000 \leq n+m$	36	13	51	100
Total	all	117	107	336	560

Table 1: CUTEr test set problem sizes and characteristics

The distribution of problem types and sizes (Very Small, Small, Medium, and Large) for our test set is shown in Table 1. We use the value $n+m$ to characterize a problem's size where n is the number of variables and m is the number of general constraints (not including bounds on the variables).

In Table 2 we summarize the number (# Opt) and percentage (% Opt) of problems for which each solver reported finding the optimal solution, discriminated by problem characteristics. On 7 problems SNOPT terminates with the message "optimal, but the requested accuracy could not be achieved" which implies that SNOPT was within a factor of 10^{-2} of satisfying the convergence conditions. It is questionable whether or not to count such problems as successes for testing purposes. In practice, such a message is very useful, however, both SLIQUE and KNITRO report any problem for which it cannot meet the desired accuracy

Problem class	Sample size	Slique # Opt	Slique % Opt	Knitro # Opt	Knitro % Opt	Snopt # Opt	Snopt % Opt
QP	117	100	85.5	113	96.6	99	84.6
BC	107	97	90.7	98	91.6	71	66.3
GC	336	259	77.1	295	87.8	285	84.8
Total	560	456	81.4	506	90.4	455	81.3

Table 2: Robustness results by problem class

in the stopping condition as a failure, even if it comes very close and it is suspected that the iterate has converged to a locally optimal point. Therefore, in order to be consistent, we do not count these problems as successes for Snopt. Since the number of such problems is small relatively speaking, their overall effect is negligible.

Even though Slique is significantly less robust than the solver Knitro on this problem set, it is similar to Snopt in terms of overall robustness. We find this encouraging since many features of our software implementation can be improved, as discussed in the final section of this paper.

Next we compare in Table 3 the robustness of the solvers based on problem size. Note the decrease in reliability of Slique as the problem size varies from medium (M) to large (L). Included in the failures for Slique are ten large QPs in which Slique (but not the other codes) experienced difficulties with memory and could not run properly. Out of the remaining 27 failures for Slique on the large set, 24 of them result from reaching the CPU limit. Clearly, for large-scale problems the current implementation of Slique is too slow. Some of the reasons for this will be discussed later on. Snopt also struggles on the set of large problems since many of these problems have a large reduced space leading to expensive computations involving a dense reduced Hessian matrix.

Problem class	Sample size	Slique # Opt	Slique % Opt	Knitro # Opt	Knitro % Opt	Snopt # Opt	Snopt % Opt
VS	266	251	94.4	251	94.4	250	94.0
S	76	52	68.4	67	88.2	70	92.1
M	118	90	76.3	103	87.3	83	70.3
L	100	63	63.0	85	85.0	52	52.0
Total	560	456	81.4	506	90.4	455	81.3

Table 3: Robustness results by problem size

11.2 Function Evaluations and Time

We now study the performance of Slique, Knitro and Snopt based on number of function/constraint evaluations and total CPU time required to achieve convergence. Our primary interest is in gauging the efficiency of the SLP-EQP approach on medium-scale and large-scale problems. For this reason, in this section we will restrict ourselves to only those problems in our test set for which $n + m \geq 1000$.

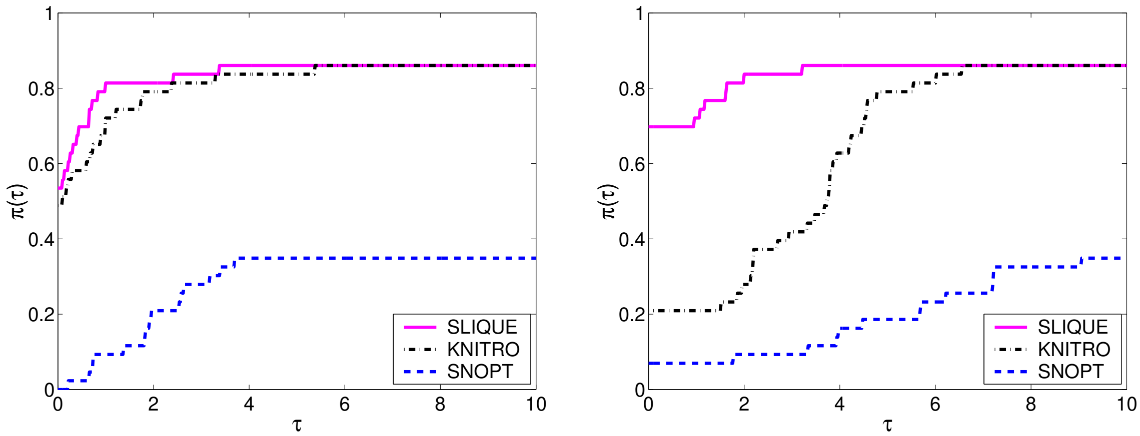
For the number of function/constraint evaluations we take the maximum of these two quantities. In order to ensure that the timing results are as accurate as possible, all tests involving timing were carried out on a dedicated machine with no other jobs running.

All the results in this section will be presented using the performance profiles proposed by Dolan and Moré [7]. In the plots $\pi_s(\tau)$ denotes the logarithmic performance profile

$$\pi_s(\tau) = \frac{\text{no. of problems where} \log_2(r_{p,s}) \leq \tau}{\text{total no. of problems}}, \qquad \tau \geq 0, \tag{11.3}$$

where $r_{p,s}$ is the ratio between the time to solve problem p by solver s over the lowest time required by any of the solvers. The ratio $r_{p,s}$ is set to infinity (or some sufficiently large number) whenever solver s fails to solve problem p. See [7] for more details on the motivation and definition of the performance profiles.

First, we compare in Figures 1(a) and 1(b) the performance of the three codes on 43 problems whose only constraints are simple bounds on the variables. Although there exist specialized approaches for solving these types of problems [6, 17, 24], it is instructive to observe the performance of SLIQUE when the feasible region has the geometry produced by simple bounds. Figures 1(a) and 1(b) indicate that SLIQUE performs quite well on this class of problems.

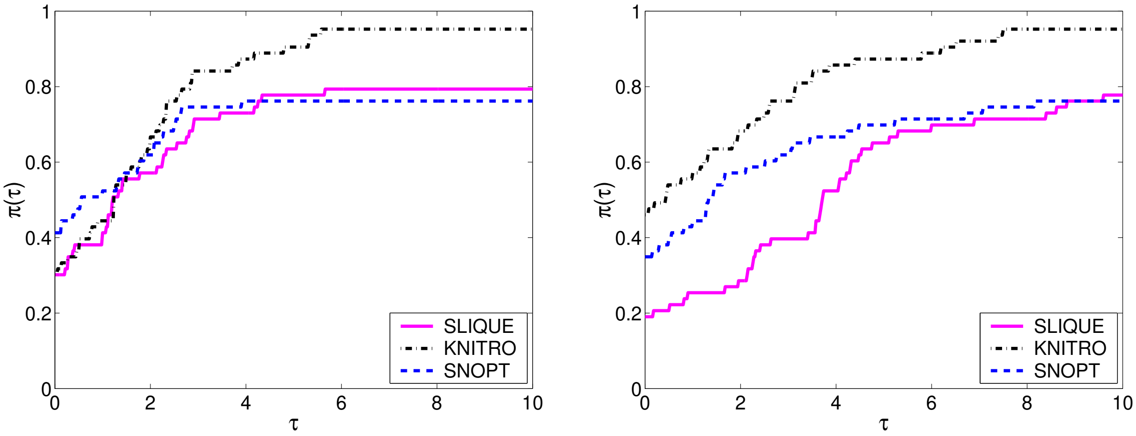


(a) Function evaluations. (b) CPU time.

Figure 1: Medium and Large Bound Constrained Problems.

Next, we compare the performance of SLIQUE, KNITRO and SNOPT on 63 quadratic programming problems from the CUTEr collection where $n + m \geq 1000$. We have excluded QPs which only have equality constraints. There are both convex and nonconvex QPs in this set. We compare these codes in terms of number of function/constraint evaluations and CPU time in Figures 2(a) and 2(b).

Note that SLIQUE is similar to the other solvers in terms of function evaluations on this set (although KNITRO is more robust), but it is less efficient in terms of CPU time. This is



(a) Function evaluations.

(b) CPU time.

Figure 2: Medium and Large Quadratic Programming Problems.

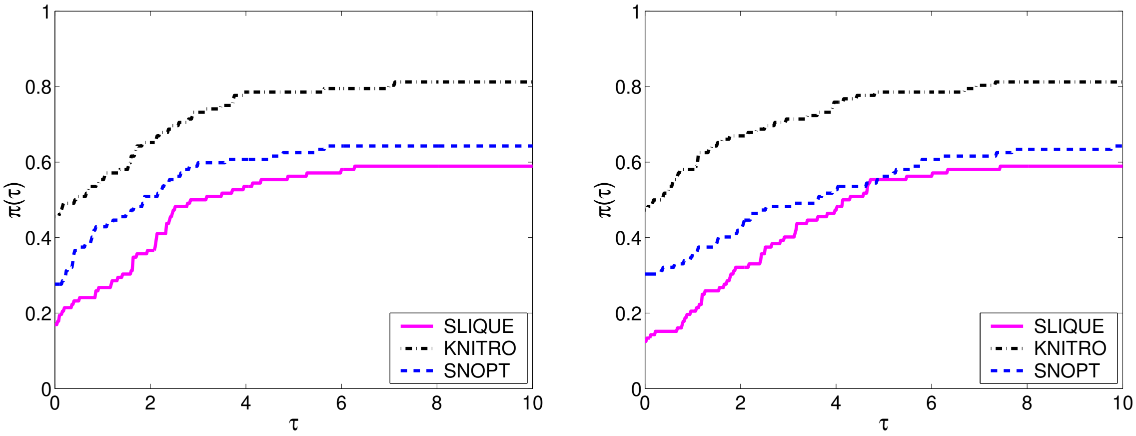
a bit surprising. We would expect that if SLIQUE is similar to SNOPT in terms of number of function evaluations, that it would also be comparable or perhaps more efficient in terms of time, since in general we expect an SLP-EQP iteration to be cheaper than an active-set SQP iteration (and typically the number of function evaluations is similar to the number of iterations). In many of these cases, the average number of inner simplex iterations of the LP solver per outer iteration in SLIQUE greatly exceeds the average number of inner QP iterations per outer iteration in SNOPT. This is caused, in part, by the inability of the current implementation of SLIQUE to perform effective warm starts, as will be discussed in Section 11.3.

Finally we consider the performance of the three codes on 112 generally constrained problems. In Figures 3(a) and 3(b), we report results for the medium-scale and large-scale generally constrained (GC) set. As in the set of quadratic programs the interior-point code KNITRO outperforms both active-set codes. Comparing the two active-set codes, SLIQUE is a little behind SNOPT, in terms of both function evaluations and CPU time.

11.3 SLIQUE Timing Statistics and Conclusions

We present below some more detailed statistics on the performance of SLIQUE on the CUTEr set of test problems. In Tables 4 and 5 we look at the average percentage of time spent on various tasks in SLIQUE based on problem characteristics and problem size respectively. These average values are obtained by computing the percentages for all the individual problems and then averaging these percentages over all the problems in the test set, where all problems are given equal weight. In this way, problems which take the most time do not dominate the timing statistics.

In these timing statistics we only include problems in which an optimal solution was found and for which the total CPU time was at least one second. We look at the following tasks: the solution of the LP subproblem (% LP); the solution of the EQP subproblem



(a) Function evaluations.

(b) CPU time.

Figure 3: Medium and Large Generally Constrained Problems.

(%EQP); the time spent factoring the augmented system matrix (i.e., the coefficient matrix in (5.9)) (% AugFact); the time spent evaluating the functions, gradients and Hessian (% Eval); and all other time (% Other).

Prob. class	% LP	% EQP	% AugFact	% Eval	% Other
QP	62.7	20.1	5.5	4.5	7.2
BC	5.3	57.8	3.9	23.5	9.5
GC	43.2	21.8	10.2	12.6	12.2
Total	40.5	29.7	7.3	12.6	10.0

Table 4: SLIQUE timing results by problem class. Average percentage of time spent on various tasks.

It is apparent from these tables that, in general, the solution of the LP subproblems dominates the overall cost of the algorithm with the solution of the EQP being the second most costly feature. An exception is the class of bound constrained problems where the computational work is dominated by the EQP phase. It is also clear that the LP cost dominates the overall cost of the algorithm more and more as the problem size grows.

Upon further examination, it is clear that there are two sources for the excessive LP times. For some problems, the first few iterations of SLIQUE require a very large number of simplex steps. On other problems, the number of LP iterations does not decrease substantially as the solution of the nonlinear program is approached, i.e., the warm start feature is not completely successful. Designing an effective warm start technique for our SLP-EQP approach is a challenging research question, since the set of constraints active at the solution of the LP subproblem often include many trust-region constraints which may change from one iteration to the next even when the optimal active set for the NLP is

Problem size	% LP	% EQP	% AugFact	% Eval	% Other
$1 \leq n+m < 100$	10.3	16.2	4.8	47.9	20.9
$100 \leq n+m < 1000$	35.9	16.4	13.2	19.2	15.3
$1000 \leq n+m < 10000$	42.5	33.1	7.7	7.5	9.2
$10000 \leq n+m$	48.8	34.7	5.0	5.9	5.5
Total	40.5	29.7	7.3	12.6	10.0

Table 5: SLIQUE timing results by problem size. Average percentage of time spent on various tasks.

identified. In contrast, warm starts are generally effective in SNOPT for which the number of inner iterations decreases rapidly near the solution.

We conclude this section by making the following summary observations about the algorithm, based on the tests reported here; see also [21].

- SLIQUE is currently quite robust and efficient for small and medium-size problems. It is very effective for bound constrained problems of all sizes, where the LP is much less costly.

- The strategy for updating the penalty parameter ν in SLIQUE has proved to be effective. Typically it chooses an adequate value of ν quickly and keeps it constant thereafter (in our tests, roughly 90% of the iterations used the final value of ν, and ν was increased about once per problem on the average). Therefore, the choice of the penalty parameter does not appear to be a problematic issue in our approach.

- The active set identification properties of the LP phase are, generally, effective. This is one of the most positive observations of this work. Nevertheless, in some problems SLIQUE has difficulties identifying the active set near the solution, which indicates that more work is needed to improve our LP trust region update mechanism.

- The active-set codes, SLIQUE and SNOPT are both significantly less robust and efficient for large-scale problems overall, compared to the interior-point code KNITRO. It appears that these codes perform poorly on large problems for different reasons. The SQP approach implemented by SNOPT is inefficient on large-scale problems because many of these have a large reduced space leading to high computing times for the QP subproblems, or resulting in a large number of iterations due to the inaccuracy of the quasi-Newton approximation. However, a large reduced space is not generally a difficulty for SLIQUE (as evidenced by its performance on the bound constrained problems).

  By contrast, the SLP-EQP approach implemented in SLIQUE becomes inefficient for large-scale problems because of the large computing times in solving the LP subproblems, and because warm starting these LPs can sometimes be ineffective. Warm starts in SNOPT, however, appear to be very efficient.

12 Final Remarks

We have presented a new active-set, trust-region algorithm for large-scale optimization. It is based on the SLP-EQP approach of Fletcher and Sainz de la Maza. Among the novel features of our algorithm we can mention: (i) a new procedure for computing the EQP step using a quadratic model of the penalty function and a trust region; (ii) a dogleg approach for computing the total step based on the Cauchy and EQP steps; (iii) an automatic procedure for adjusting the penalty parameter using the linear programming subproblem; (iv) a new procedure for updating the LP trust-region radius that allows it to decrease even on accepted steps to promote the identification of locally active constraints.

The experimental results presented in Section 11 indicate, in our opinion, that the algorithm holds much promise. In addition, the algorithm is supported by the global convergence theory presented in [2], which builds upon the analysis of Yuan [23].

Our approach differs significantly from the SLP-EQP algorithm described by Fletcher and Chin [4]. These authors use a filter for step acceptance. In the event that the constraints in the LP subproblem are incompatible, their algorithm solves instead a feasibility problem that minimizes the violation of the constraints while ignoring the objective function. We prefer the ℓ_1-penalty approach (3.4) because it allows us to work simultaneously on optimality and feasibility, but testing would be needed to establish which approach is preferable. The algorithm of Fletcher and Chin defines the trial step to be either the full step to the EQP point (plus possibly a second order correction) or, if this step is unacceptable, the Cauchy step. In contrast, our approach explores a dogleg path to determine the full step. Our algorithm also differs in the way the LP trust region is handled and many other algorithmic aspects.

The software used to implement the SLIQUE algorithm is not a finished product but represents the first stage in algorithmic development. In our view, it is likely that significant improvements in the algorithm can be made by developing: (i) faster procedures for solving the LP subproblem, including better initial estimates of the active set; (ii) improved strategies for updating the LP trust region; (iii) an improved second-order correction strategy or a replacement by a non-monotone strategy; (iv) preconditioning techniques for solving the EQP step; (v) mechanisms for handling degeneracy.

References

[1] 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.

[2] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. On the convergence of successive linear programming algorithms. Technical Report OTC 2002/5, Optimization Technology Center, Northwestern University, Evanston, IL, USA, 2002.

[3] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[4] C. M. Chin and R. Fletcher. On the global convergence of an SLP-filter algorithm that takes EQP steps. Numerical Analysis Report NA/199, Department of Mathematics, University of Dundee, Dundee, Scotland, 1999.

[5] A. R. Conn, N. I. M. Gould, and Ph. Toint. *Trust-region methods.* MPS-SIAM Series on Optimization. SIAM publications, Philadelphia, Pennsylvania, USA, 2000.

[6] 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.

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

[8] R. Fletcher. *Practical Methods of Optimization. Volume 2: Constrained Optimization.* J. Wiley and Sons, Chichester, England, 1981.

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

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

[11] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 2002.

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

[13] N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic problems arising in optimization. *SIAM Journal on Scientific Computing*, 23(4):1375–1394, 2001.

[14] N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999.

[15] Harwell Subroutine Library. *A catalogue of subroutines (HSL 2000).* AEA Technology, Harwell, Oxfordshire, England, 2002.

[16] ILOG CPLEX 8.0. *User's Manual.* ILOG SA, Gentilly, France, 2002.

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

[18] N. Maratos. *Exact penalty function algorithms for finite-dimensional and control optimization problems.* PhD thesis, University of London, London, England, 1978.

[19] M. J. D. Powell. A Fortran subroutine for unconstrained minimization requiring first derivatives of the objective function. Technical Report R-6469, AERE Harwell Laboratory, Harwell, Oxfordshire, England, 1970.

[20] M. J. D. Powell. A new algorithm for unconstrained optimization. In J. B. Rosen, O. L. Mangasarian, and K. Ritter, editors, *Nonlinear Programming*, pages 31–65, London, 1970. Academic Press.

[21] R. A. Waltz. *Algorithms for large-scale nonlinear optimization*. PhD thesis, Department of Electrical and Computer Engineering, Northwestern University, Evanston, Illinois, USA, http://www.ece.northwestern.edu/˜rwaltz/, 2002.

[22] R. A. Waltz and J. Nocedal. KNITRO user's manual. Technical Report OTC 2003/05, Optimization Technology Center, Northwestern University, Evanston, IL, USA, April 2003.

[23] Y. Yuan. Conditions for convergence of trust region algorithms for nonsmooth optimization. *Mathematical Programming*, 31(2):220–228, 1985.

[24] 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.