A COMPARATIVE STUDY OF LARGE-SCALE NONLINEAR OPTIMIZATION ALGORITHMS

HANDE Y. BENSON, DAVID F. SHANNO, AND ROBERT J. VANDERBEI

Operations Research and Financial Engineering
Princeton University
ORFE-01-04

Revised July 17, 2002

Abstract. In recent years, much work has been done on implementing a variety of algorithms in nonlinear programming software. In this paper, we analyze the performance of several state-of-the-art optimization codes on large-scale nonlinear optimization problems. Extensive numerical results are presented on different classes of problems, and features of each code that make it efficient or inefficient for each class are examined.

1. Introduction

Real-world optimization problems are often large and nonlinear. A financial optimization problem can have hundreds of thousands of variables and constraints. Also, many engineering problems arise as discretizations of differential equations and can be solved more accurately as their discretizations become finer and the problem size larger. These are only two examples of problems where the modelers would like to solve the largest possible problems they can to obtain the best possible solutions.

Date: July 17, 2002.

Key words and phrases. interior-point methods, large-scale optimization, nonlinear programming.

Research of the first and third authors supported by NSF grant DMS-9870317, ONR grant N00014-98-1-0036. Research of the second author supported by NSF grant DMS-0107450.

Computing advances in recent years have made it possible for researchers in optimization to develop tools to solve such large problems. Computer memory is cheap, and processors are fast. Many efficient data structures and algorithms have been developed. Modeling languages themselves have evolved, as well, making it easier to express the problem. Some of these languages, including AMPL, which is described in [8] and is used as the modeling language in this paper, provide automatic differentiation capabilities for first and second derivatives.

Our concern is to solve "large-scale" NLPs. We will define the size of a problem to be the number of variables plus the number of constraints, or $n+m$. Large-scale problems we consider have sizes of at least 1000.

In this paper, we present and test some of the state-of-the-art codes available for solving large-scale NLPs. Our goal is to identify features of these codes that are efficient and those that are inefficient on a variety of problem classes. We have worked with three algorithms:

- LOQO, an interior-point method code by Vanderbei et. al. [15]
- KNITRO, a trust-region algorithm by Byrd et. al. [2]
- SNOPT, a quasi-Newton algorithm by Gill et. al. [9]

A fourth code, LANCELOT by Conn et. al. [4], is designed for large-scale nonlinear optimization, but previous work with the code [6] has shown it not competitive with the above codes for the larger problems tested. Thus, we do not include any further discussion of it here. A fifth code, FILTER by Fletcher and Leyffer [7], is relatively new—it shows great promise on smaller problems. However, it currently does not have a sparse implementation and encounters memory allocation problems on large-scale models.

Comparison of various aspects of the algorithms under study leads to a number of conclusions concerning specific algorithmic details, which we summarize briefly here. For problems with large degrees of freedom, methods using second order information significantly outperform methods using only first-order information. Direct matrix factorizations of a Newton matrix are more efficient than indirect solution methods given sufficient sparsity in the factorization, while indirect methods can be superior for dense factorization of well-conditioned matrices. The relative efficiency of active set methods as opposed to interior-point methods in selecting the active constraints remains open. Finally, specific algorithmic details for specialized problems are discussed.

2. LOQO: An Infeasible Interior-Point Method

We begin with a description of the LOQO algorithm. The basic problem we consider is

$$\begin{array}{lll} \text{minimize} & f(x) & \\ \text{subject to} & h_i(x) \geq 0, & i = 1, \ldots, m \end{array} \tag{1}$$

where $x \in \mathbb{R}^n$ are the decision variables, $f : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}^m$. Since the algorithm involves derivatives, $f(x)$ and the $h_i(x)$'s are assumed to be twice continuously differentiable.

Expressing nonlinear optimization problems in the form given by (1) is rather restrictive—the actual implementation of LOQO allows for much greater variety in the form of expression, including bounds on variables, ranges on inequalities, and equality constraints. We will discuss the case of equality constraints in Section 5. For further details on the general case, see [14].

LOQO starts by adding slacks to the inequality constraints in (1), which becomes

$$\begin{array}{c} \text{minimize } f(x) \\ \text{subject to } h(x) - w = 0, \qquad w \geq 0, \end{array} \tag{2}$$

where $h(x)$ and w are vectors representing the $h_i(x)$'s and w_i's, respectively. The inequality constraints are then eliminated by incorporating them in a logarithmic barrier term in the objective function, $b_\mu(x, w) = f(x) - \mu \sum_{i=1}^m \log(w_i)$, transforming (2) to

$$\begin{array}{c} \text{minimize } b_\mu(x, w) \\ \text{subject to } h(x) - w = 0. \end{array} \tag{3}$$

Denoting the Lagrange multipliers for the system (3) by y, the first order conditions for the problem are

$$\begin{aligned} \nabla f(x) - \nabla h(x) y &= 0 \\ -\mu W^{-1} e + y &= 0 \\ h(x) - w &= 0 \end{aligned} \tag{4}$$

where W is the diagonal matrix with the w_i's as diagonal elements, e is the vector of all ones, and $\nabla h(x)$ is the transpose of the Jacobian matrix of the vector $h(x)$. The primal-dual system is obtained from (4) by multiplying the second equation by W, giving the system

$$\begin{aligned} \nabla f(x) - \nabla h(x)^T y &= 0 \\ -\mu e + WYe &= 0 \\ h(x) - w &= 0 \end{aligned} \tag{5}$$

where Y is the diagonal matrix with the y_i's on the diagonal.

Newton's method is employed to iterate to a triple (x, w, y) satisfying (5). Letting

$$H(x,y) = \nabla^2 f(x) - \sum_{i=1}^{m} y_i \nabla^2 h_i(x)$$

and

$$A(x) = \nabla h(x)^T,$$

the Newton system for (5) is then

$$\begin{bmatrix} H(x,y) & 0 & -A(x)^T \\ 0 & Y & W \\ A(x) & -I & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta y \end{bmatrix} = \begin{bmatrix} -\nabla f(x) + A(x)^T y \\ \mu e - WYe \\ -h(x) + w \end{bmatrix}.$$

This system is not symmetric, but can be symmetrized by negating the first equation and multiplying the second by $-W^{-1}$, giving the system

$$\begin{bmatrix} -H(x,y) & 0 & A(x)^T \\ 0 & -W^{-1}Y & -I \\ A(x) & -I & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta y \end{bmatrix} = \begin{bmatrix} \sigma \\ -\gamma \\ \rho \end{bmatrix}, \tag{6}$$

where

$$\begin{aligned} \sigma &= \nabla f - A(x)^T y, \\ \gamma &= \mu W^{-1} e - y, \\ \rho &= w - h(x). \end{aligned}$$

Here, σ, γ, and ρ depend on x, y, and w, even though we do not show this dependence explicitly in our notation. Note that ρ measures *primal infeasibility*, and using an analogy with linear programming, we refer to σ as the *dual infeasibility*.

It is easy to eliminate Δw from this system without producing any additional fill-in in the off-diagonal entries. Thus, Δw is given by

$$\Delta w = WY^{-1}(\gamma - \Delta y).$$

After the elimination, the resulting set of equations is the *reduced KKT system*:

$$\begin{bmatrix} -H(x,y) & A(x)^T \\ A(x) & WY^{-1} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} \sigma \\ \rho + WY^{-1}\gamma \end{bmatrix}. \tag{7}$$

This system is solved using modified Cholesky factorization and a backsolve step. Since such a procedure benefits from sparsity, a symbolic factorization is performed first and an appropriate permutation to improve the sparsity structure of the Cholesky factor of the reduced

KKT matrix is found. There are two symbolic factorization heuristics available in LOQO: Multiple Minimum Degree and Minimum Local Fill. The user can also turn off the symbolic factorization step altogether. The default in LOQO is Multiple Minimum Degree. The details of these heuristics are given in [12]. LOQO also provides a mechanism for assigning priorities in factoring the columns corresponding to Δx or Δy. These are called, respectively, *primal* and *dual* orderings in LOQO. They are discussed in [15].

LOQO generally solves a modification of (7) in which λI is added to $H(x, y)$. For $\lambda = 0$, we have the Newton directions. As $\lambda \to \infty$, the directions approach the steepest descent directions. Choosing λ so that $H(x, y) + A(x)^T W^{-1} Y A(x) + \lambda I$ is positive definite ensures that the step directions are descent directions. See [15] for further details.

Having computed step directions, Δx, Δw, and Δy, LOQO then proceeds to a new point by

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \alpha^{(k)} \Delta x^{(k)}, \\ w^{(k+1)} &= w^{(k)} + \alpha^{(k)} \Delta w^{(k)}, \\ y^{(k+1)} &= y^{(k)} + \alpha^{(k)} \Delta y^{(k)}, \end{aligned} \tag{8}$$

where $\alpha^{(k)}$ is chosen to ensure that $w^{(k+1)} > 0$, $y^{(k+1)} > 0$, and either the barrier function, b_μ, or the infeasibility, $\|\rho\|$ is reduced. (Here, and throughout the paper, all norms are Euclidean, unless otherwise indicated). A sufficient reduction is determined by an Armijo rule. Details of this "filter" approach to LOQO are discussed in [1].

3. KNITRO: A Trust-Region Algorithm

KNITRO [2] is an interior-point solver developed by Byrd et. al. It takes a barrier approach to the problem using sequential quadratic programming (SQP) and trust regions to solve the barrier subproblems at each iteration. The SQP technique is used to handle the nonlinearities in the problem and the trust region strategy serves to handle both convex and nonconvex problems.

KNITRO solves the first-order optimality conditions given by (4) approximately, by forming a quadratic model to approximate the Lagrangian function. A step $(\Delta x, \Delta w)$ is computed at each iteration by minimizing this quadratic model subject to a linear approximation to the constraints:

$$\begin{aligned} \min \quad & \nabla f(x)^T \Delta x + \tfrac{1}{2} \Delta x^T H \Delta x - \mu e^T W^{-1} \Delta w + \tfrac{1}{2} \Delta w^T W^{-1} Y \Delta w \\ \text{s.t.} \quad & A\Delta x - \Delta w - \rho = 0 \\ & \|(\Delta x, W^{-1} \Delta w)\| \le \delta, \end{aligned} \tag{9}$$

where δ is the trust-region radius. This problem is solved using a two-phase method consisting of a vertical step to satisfy the linear constraints and then a horizontal step to improve optimality.

The vertical step computes the best possible step to attempt to satisfy the linearized constraints. This step is the "best" in the least-squares sense. They compute the vertical step, $v = (v_x, v_w)$, as a solution to the following problem:

$$\begin{array}{ll} \text{minimize} & \|Av_x - v_w + \rho\|^2 \\ & \|v_x - W^{-1}v_w\| \leq \xi\Delta, \end{array}$$

where ξ is a real parameter typically set to 0.8.

Letting

$$\tilde{v} = \begin{bmatrix} v_x \\ W^{-1}v_w \end{bmatrix}, \text{ and}$$

$$\hat{A} = \begin{bmatrix} A & -W \end{bmatrix}, \tag{10}$$

this problem can be rewritten as

$$\begin{array}{ll} \text{minimize} & -2\rho^T\hat{A}\tilde{v} + \tilde{v}^T\hat{A}^T\hat{A}\tilde{v} \\ & \|\tilde{v}\| \leq \xi\Delta. \end{array}$$

Then,

$$\tilde{v} = \hat{A}^T(\hat{A}^T\hat{A})^{-1}\rho, \tag{11}$$

that is $\tilde{v}$ is computed by solving an augmented system. KNITRO takes a truncated Newton step to satisfy the bounds on the slacks. Finally, v is obtained by computing $(\tilde{v}_x, W\tilde{v}_w)$.

Once the vertical step is computed, the problem given by (9) can be rewritten in terms of the scaled horizontal step, $\tilde{s}$, as

$$\begin{array}{ll} \text{minimize} & \nabla f(x)^T\tilde{s}_x - \mu e^T\tilde{s}_w + \tilde{v}^T\hat{H}^Ts + \frac{1}{2}\tilde{s}^T\hat{H}\tilde{s} \\ \text{subject to} & \hat{A}\tilde{s} = 0, \\ & \|\tilde{s}\|^2 \leq \Delta^2 - \|\tilde{v}\|^2 \end{array}$$

where

$$\hat{H} = \begin{bmatrix} H & 0 \\ 0 & YW \end{bmatrix}.$$

Let Z be a basis for the null space of $\hat{A}$. Since $\tilde{s}$ is constrained to lie in the null space of $\hat{A}$, we have that

$$\tilde{s} = Zu$$

for some $u \in \mathbb{R}^n$. Thus, we can further simplify the QP subproblem as

$$\begin{array}{ll} \text{minimize} & \begin{bmatrix} \nabla f(x)^T & -\mu e^T \end{bmatrix} Zu + \tilde{v}^T\hat{H}^TZu + \frac{1}{2}u^TZ^T\hat{H}Zu \\ \text{subject to} & \|Zu\|^2 \leq \Delta^2 - \|\tilde{v}\|^2. \end{array}$$

KNITRO solves this problem using a conjugate gradient method.

Note that in finding the step directions, there are trust region constraints associated with each subproblem. The purpose of the trust region constraints is discussed in detail in [2]. We note that the trust region constraint in the horizontal problem has an effect similar to LOQO's diagonal perturbation of the reduced KKT matrix.

After obtaining the step $(\Delta x, \Delta w)$ from the vertical and the horizontal steps, KNITRO checks to see if it provides a sufficient reduction in the merit function

$$\phi(x, w, \beta) = b_\mu(x, w) + \beta\|\rho\|.$$

If it does not, the step is rejected and the trust region radius is reduced. If the reduction is better than the "predicted reduction," then the trust region radius is increased.

3.1. **Lagrange Multipliers.** There is a difference between LOQO's and KNITRO's algorithms in the computation of the Lagrange multipliers. In LOQO, steps in both primal and dual variables are obtained from the solution of the system (7). In KNITRO, however, the Lagrange multipliers depend on the primal variables (x, w) and are computed at each iteration as follows:

$$y = (\hat{A}\hat{A}^T)^{-1}\hat{A}\begin{bmatrix}\nabla f(x)\\ -\mu e\end{bmatrix}.$$

4. SNOPT: An Active-Set Approach

SNOPT [9] is an SQP algorithm that uses an active-set approach. It is a first-order code, employing a limited-memory quasi-Newton approximation for the Hessian of the Lagrangian.

SNOPT uses a *modified* Lagrangian function:

$$L_m(x, x^{(k)}, y^{(k)}) = f(x) - y^T h_L(x, x^{(k)}),$$

where

$$h_L(x, x^{(k)}) = h(x) - h(x^{(k)}) - A(x^{(k)})^T(x - x^{(k)})$$

is the departure from linearity of the constraint functions. The algorithm seeks a stationary point to the Lagrangian through solving a sequence of quadratic approximations. Each quadratic subproblem can be expressed as

(12)

$$\begin{array}{ll}\text{minimize} & f(x^{(k)}) + \nabla f(x^{(k)})^T(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T\tilde{H}(x^{(k)})(x - x^{(k)})\\ \text{subject to} & h(x^{(k)}) + A(x^{(k)})^T(x - x^{(k)}) - w = 0\\ & w \geq 0,\end{array}$$

where $w \in \mathbb{R}^m$ are the slack variables and $\tilde{H}(x^{(k)})$ is an approximation to the Hessian of the Lagrangian obtained using the limited-memory BFGS method.

Problem (12) is solved using a two-phase method. Phase 1 is the *feasibility phase*, wherein the following problem is solved:

$$\begin{array}{ll} \text{minimize} & e^T(v+s) \\ \text{subject to} & h(x^{(k)}) + A(x^{(k)})^T(x - x^{(k)}) - w = 0 \\ & w - v + s \geq 0 \\ & w \geq 0, v \geq 0, s \geq 0. \end{array}$$

The solution of this problem is the minimum total constraint violation. If the problem (12) is feasible, there exists a solution with v and s equal to 0. The values of x and w at the solution are used as the starting point for Phase 2, which is called the *optimality phase*.

In order to describe Phase 2, the concept of a *working set* needs to be introduced. The working set is the current prediction of the constraints and bounds that hold with equality at optimality. Such constraints and bounds are said to be *active*. The slacks associated with these constraints are called *nonbasic* variables—they are set to 0. The working set at optimality is called the *active set*. Let V denote the rows of the matrix

$$\begin{bmatrix} A & -I \\ 0 & I \end{bmatrix}$$

that correspond to the working set. The search direction $(\Delta x, \Delta w)$ is taken to keep the working set unaltered, so that

$$V \begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix} = 0.$$

Let Z be the full-rank matrix that spans the null-space of v. If the working set (or a permutation thereof) is partitioned into

$$V = \begin{bmatrix} B & S & N \\ 0 & 0 & I \end{bmatrix},$$

then, the null-space basis (subject to the same permutation) is given by

$$Z = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix}.$$

Here B denotes the square matrix corresponding to the basic variables, S denotes the matrix corresponding to the superbasic variables, and N denotes the matrix corresponding to the nonbasic variables.

Using Z, the *reduced gradient*, ∇f_z, and the first-order approximation to the *reduced Hessian*, $\tilde{H}_z$, are defined as follows:

$$\nabla f_z = Z^T \nabla f, \qquad \tilde{H}_z = Z^T \tilde{H} Z.$$

If the reduced gradient is zero at the point (x, s) computed at the end of Phase 1, then a stationary point of (12) has been reached. Otherwise, a search direction $(\Delta x, \Delta w)$ is found so that

$$\tilde{H}_z \begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix} = -\nabla f_z.$$

The solution of this system is obtained using Cholesky factorization.

Because of its active set approach, SNOPT works well when the degrees of freedom, that is $n - m$, is small, generally in the several hundreds. Furthermore, the code differentiates between linear and nonlinear variables and reduces the size of the Hessian accordingly, so an NLP with large numbers of linear constraints and variables is ideal for SNOPT.

5. Other Algorithmic Details

5.1. **Equality Constraints.** Consider the NLP with equality constraints.

$$\begin{array}{lllll} \text{minimize} & f(x) & & & \\ \text{subject to} & h_i(x) & \geq & 0, & i \in \mathcal{I} \\ & g_i(x) & = & 0, & i \in \mathcal{E}. \end{array} \tag{13}$$

LOQO converts the equality constraints into inequalities by considering them as range constraints of the form

$$0 \leq g_i(x) \leq r_i,$$

where the range, r_i, is equal to 0. Each range constraint, then, becomes

$$\begin{aligned} g_i(x) - w_i &= 0, \\ w_i + p_i &= r_i, \\ w_i, p_i &\geq 0. \end{aligned}$$

This formulation with $r_i = 0$ ensures that the slacks w_i and p_i vanish at optimality and $g_i(x) = 0$ there, too. Note that we are not breaking the equality constraint into two inequalities, thus avoiding the problem of introducing singularity into the KKT matrix. However, for each equality constraint, we now have an additional 2 slack variables which start at a default value (generally 1) and must be iteratively brought down to 0.

Unlike LOQO, KNITRO does not add slacks to the equality constraints of the problem given by (13). Therefore, the only change to the algorithm presented above is the redefinition of $\hat{A}$:

$$\hat{A} = \begin{bmatrix} A_g & 0 \\ A_h & -I \end{bmatrix},$$

where A_g is the Jacobian of the inequality constraints and A_h is the Jacobian of the equality constraints.

SNOPT is similar to LOQO in that equality constraints are treated as range constraints, with both the lower and upper bounds equal to 0. This changes Phase 1 slightly:

$$\begin{array}{ll} \text{minimize} & e_h^T(v_h + s_h) + e_g^T(v_g + s_g) \\ \text{subject to} & h(x^{(k)}) + A_h(x^{(k)})^T(x - x^{(k)}) - w_h = 0 \\ & g(x^{(k)}) + A_g(x^{(k)})^T(x - x^{(k)}) - w_g = 0 \\ & w_h - v_h + s_h \geq 0 \\ & w_g - v_g + s_g = 0 \\ & w_h \geq 0, v_h \geq 0, s_h \geq 0 \\ & w_g \geq 0, v_g \geq 0, s_g \geq 0, \end{array}$$

where e_h and e_g are vectors of all ones of appropriate dimensions, A_h and A_g are the Jacobians of the inequality and equality constraints, respectively, and (w_h, v_h, s_h) and (w_g, v_g, s_g) are slacks associated with the inequality and equality constraints, respectively. The rest of the algorithm proceeds without much change. The only additional point is that once a feasible solution is found in Phase 1, the equality constraints remain active.

5.2. **Free Variables.** In order to keep the reduced KKT matrix quasidefinite [13], LOQO splits free variables into a difference of two nonnegative variables. If the variable x_j is free, then two additional slacks variables are created for it:

$$\begin{array}{l} x_j - t_j + g_j = 0 \\ t_j, g_j \geq 0. \end{array}$$

These slacks are incorporated into the logarithmic barrier function, and the constraints become a part of the KKT system. For more details, see [15].

There are no modifications to KNITRO's algorithm as described in Section 3 for handling free variables.

SNOPT treats free variables as having lower and upper bounds of $-\infty$ and $+\infty$, respectively. As described above, variable bounds are treated as constraints, so SNOPT adds two slacks to the bounds, effectively

splitting the free variable. The problem, then, becomes:

$$\begin{array}{ll} \text{minimize} & e^T(v+s) + \overline{e}^T(\overline{v}+\overline{s}) \\ \text{subject to} & h(x^{(k)}) + A(x^{(k)})^T(x - x^{(k)}) - w = 0 \\ & w - v + s \geq 0 \\ & -\infty \leq x - \overline{v} + \overline{s} \leq \infty \\ & w \geq 0, v \geq 0, s \geq 0 \\ & \overline{v} \geq 0, \overline{s} \geq 0, \end{array}$$

where $\overline{v}$ and $\overline{s}$ are the slacks associated with the free variables x, and $\overline{e}$ is the vector of all ones having the appropriate dimension.

5.3. **Infeasibility/Unboundedness Detection.** When solving large-scale problems, it is especially important to detect infeasibilities and unboundedness in the problem as early as possible. SNOPT's Phase 1 detects infeasibilities at the beginning of a problem by forcing the attainment of a feasible point. If no such point is found, the algorithm enters its *elastic mode*, which is described in [9].

KNITRO and LOQO, on the other hand, are infeasible interior-point algorithms—the only way to detect infeasibility and unboundedness is by having the appropriate stopping rules. In LOQO, the implementation of a filter method facilitates the tracking of progress toward feasibility separately from progress toward optimality, which helps with infeasibility detection. For example, if the steplength, α, is greatly reduced to failing to provide sufficient improvement for the filter for a series of iterations, there is no progress being made toward feasibility. Also, since LOQO is a primal-dual code, we can measure the duality gap at each iteration and stop the algorithm if this gap is not getting smaller for a series of iterations.

6. Numerical Results

In this section, we present the numerical results of solving large-scale nonlinear optimization problems using LOQO (Version 6.0) and KNITRO (Version 1.00: 09/27/01). Since codes using Newton's Method require second partial derivatives, we have formulated our test problems in AMPL [8], which is currently the only modeling language that provides second derivatives to a solver. Both solvers were compiled with the AMPL solver interface Version 20010823.

The test problems come from 3 sources: The CUTE suite compiled by Conn, Gould, Toint [3], the COPS suite by Moré [5], and several engineering problems formulated by Vanderbei [11]. Many of the problems have been changed from their original sizes to the sizes reported in this paper so that a wide range of large-scale optimization problems

		LOQO		KNITRO		SNOPT	
	Total	p	Time	p	Time	p	Time
Equality Cons QP	9	6	105.26	9	22.67	3	2839.43
Inequality Cons QP	36	25	2258.14	35	3164.67	13	1765.51
Mixed Cons QP	40	31	9982.87	37	4038.38	25	3522.76
Uncons QP	3	3	3.46	3	7.13	1	695.27
Equality Cons NLP	33	27	1845.28	28	1454.92	12	10602.12
Inequality Cons NLP	15	12	160.76	7	235.27	8	1757.52
Mixed Cons NLP	46	27	1319.44	42	7147.32	28	20105.99
Uncons NLP	56	46	3562.14	37	2304.86	16	25074.66

TABLE 1. Number of problems solved by each solver (p) and their total runtimes in CPU seconds.

could be included in the test set. The criterion for a problem to be large-scale is that the size of the problem be at least 1000, where the size of the problem is taken to be the number of variables plus the number of constraints. In general, the test problems are much larger than this, with sizes in the tens of thousands.

In Table 1, the total number of problems in each category are presented, as well as the number of problems that each code solved. On this list, we have included only those problems for which the solvers find solutions using their default settings. For example, LOQO can be tuned to solve more problems by modifying the initialization of the slack variables. The only option changed from its default value was increasing the maximum number of superbasic variables allowed for SNOPT so that it could attempt problems where this value was more than 50.

In presenting the numerical results for individual problems, we have broken down the tables by problem characteristics. The reason is that, in general, the performance of each solver can be easily explained and predicted based on these characteristics.

Before we discuss the numerical results for each category of problems, we need to establish a methodology for comparing the algorithms. It is quite easy for categories with small numbers of problems to simply examine the corresponding table in the Appendix, but for categories such as Unconstrained NLPs, this comparison is not so easy. Therefore, for categories with large numbers of problems, we will employ *performance profiles*.

6.1. **Performance Profiles.** Recently, Dolan and Moré [6] proposed a new way to compare the performance of several algorithms running

on the same set of problems. Their approach is simply to compute an estimate of the probability that an algorithm performs within a multiple of the runtime or iteration count (or any other metric) of the best algorithm.

Assume that we are comparing the performance of n_s solvers on n_p problems. Assuming also that we are using runtime as the metric in the profile, let

$$t_{p,s} = \text{runtime required to solve problem } p \text{ by solver } s.$$

Then, the *performance ratio* of solver s on problem p is defined by

$$\rho_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : 1 \leq s \leq n_s\}}.$$

Here, we also assume that when a solver s cannot find a solution to a problem p, $\rho_{p,s} = \infty$.

In order to evaluate the performance of the algorithm on the whole set of problems, one can use the quantity

$$\phi_s(\tau) = \frac{1}{n_p}\text{size}\{p : 1 \leq p \leq n_p, \rho_{p,s} \leq \tau\}.$$

The function $\phi_s : \mathbb{R} \to [0, 1]$ is called the *performance profile* and represents the cumulative distribution function of the performance ratio.

We are now ready to examine the performance of each solver on the 8 categories of problems. For each category, we will present several models and discuss the performance of the solvers on each.

(1) *Equality Constrained QP*: It is clear from the results in Table 2 that KNITRO outperforms LOQO and SNOPT in this category. The difference between the two interior-point codes stems from the use of slack variables by LOQO in the presence of equality constraints and free variables. These slack variables provide numerical stability for problems with ill-conditioned Hessians by contributing to the diagonal of the KKT matrix. However, the large problems in our test suite are generally well-behaved, and, in the absence of slack variables, LOQO can solve them in two iterations by solving the exact Newton system and taking a full step in that direction. With slack variables, however, smaller steps are taken to keep them strictly positive, and LOQO needs several iterations to reach the optimal solution. KNITRO does not use slack variables in such cases and solves a smaller system and can generally reach the optimal solution in less time.
   - *dtoc3*: This problem is an example of how slack variables affect LOQO's performance on large-scale problems. It is a

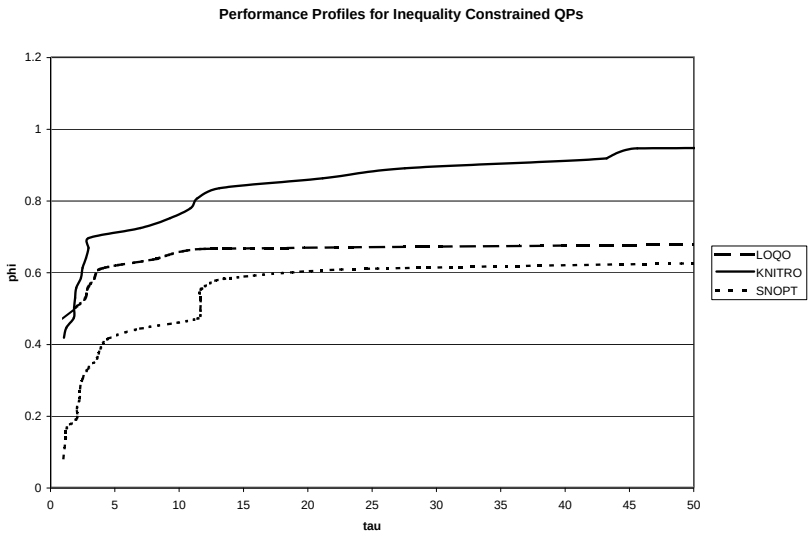

FIGURE 1. Performance Profiles for Inequality Constrained QPs.

QP with 14996 variables (all of them free) and 9997 equality constraints. LOQO solves this problem in 10 iterations and 23.83 seconds. When the slack variables are removed from the code, LOQO solves this problem in 2 iterations and 1.33 seconds. KNITRO solves this problem in 2.99 seconds, and SNOPT takes 597.32 seconds due to the many degrees of freedom.

(2) *Inequality Constrained QP*: For these problems, LOQO outperforms KNITRO and SNOPT in most cases, but it is slow or needs tuning on the *liswet* set. The performance profile for this category of problems is given in Figure 1.
   - *mosarqp1*: This problem has 2500 variables and 700 constraints. It is quite sparse, with 3422 nonzeros in the Jacobian of the constraints and 2590 nonzeros in the Hessian. Moreover, the Cholesky factor of the reduced KKT matrix has 22509 nonzeros that are well-distributed throughout the matrix, so the estimate for the number of arithmetic

operations is 703,858. LOQO, therefore, takes only 1.04 seconds to solve this problem.
KNITRO performs up to 817 conjugate gradient iterations per major iteration and takes 13.60 seconds. SNOPT spends too much time in identifying the active set, so it takes 344.92 seconds to solve the problem.

- *yao*: This problem has 2000 variables and 1999 constraints. It is quite sparse with 5996 nonzeros in the Jacobian of the constraints and 2000 nonzeros in the Hessian. LOQO, however, performs many iterations with steplength cuts to make progress toward a solution with the desired level of accuracy. The steplength cuts result in an increased number of iterations, which increases the runtime. LOQO's runtime for this problem is 17.23 seconds.
  KNITRO works with a sparse Jacobian and does up to 42 conjugate gradient iterations per major iteration. It takes 1.41 seconds to solve the problem. SNOPT does very well with this problem, solving it in only 0.13 seconds, because there are 2000 variables and 1999 constraints in this problem. SNOPT favors problems with small degress of freedom, and this problem has only one degree of freedom. However, it also ends up at a worse solution than the other two solvers.
- *cvxbqp1*: This problem has 10000 variables with bounds and no constraints. The Hessian of the objective function has 69968 nonzeros, and LOQO provides a factorization with 328,879 nonzeros and with some dense columns. In this case, the number of arithmetic operations required to factor the Hessian is 33,936,122. Not surprisingly, LOQO takes too much time, 40.88 seconds, for the 17 iterations needed to solve this problem. Only about 1 second of this time is spent in function evaluations—the rest can be attributed to the Cholesky factorization.
  KNITRO performs quite well on this problem, performing only between 1 and 11 conjugate gradient iterations per major iteration. It solves the problem in 4.96 seconds.
  SNOPT suffers from the many degrees of freedom in this unconstrained problem. It takes 10000 iterations and 121.86 seconds to solve this problem.
- *biggsb1*: This problem has 1000 variables and only 2998 nonzeros in its Hessian. LOQO finds a very sparse factorization with only 999 nonzeros in the Cholesky factor and

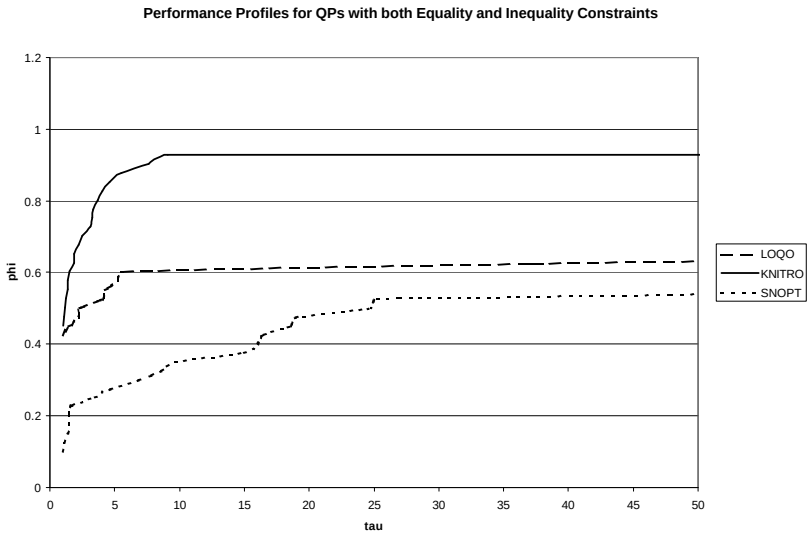

FIGURE 2. Performance Profiles for QPs with both Equality and Inequality Constraints.

hence solves this problem in 0.43 seconds. KNITRO, on the other hand, performs up to 1469 conjugate gradient iterations per major iteration and takes 18.57 seconds. SNOPT stops when it reaches its iteration limit.

(3) *QPs with both Equality and Inequality Constraints*: There are several large sets of problems in this category, and the results of the comparison are mixed as can be seen in the performance profile in Figure 2. While KNITRO solves more problems with the defaults on this category than the other two solvers, LOQO is able to solve the whole set with proper tuning of the slack initialization parameter.

- *cvxqp3*: The Jacobian of the constraints, A, and the Hessian of the Lagrangian, H are both quite sparse, but the reduced KKT matrix has a dense factorization. Therefore, LOQO takes 6173.21 seconds to solve the problem due to the time required for the Cholesky factorization.

KNITRO does not need to factor the Hessian, as it solves for the horizontal step using a conjugate gradient method. Here the matrix vector multiplications are sparse and hence efficient. The major advantage for KNITRO, however, is that it takes only one to four conjugate gradient iterations at each step, which is remarkable for a system with 10,000 variables. These two factors allow KNITRO to solve the problem in 5.60 seconds.

This problem has 10000 variables and 7500 constraints, and with so many degrees of freedom, SNOPT takes 417.44 seconds to solve it.

- *gridneta*: Sparsity of the reduced KKT matrix (and its Cholesky factor) is essential for LOQO's success on large-scale problems. The number of arithmetic operations required to factor the reduced KKT matrix at each iteration is proportional to the sum of the squares of the number of nonzeros in each column of the factor. Therefore, dense columns in the Cholesky factor mean that the number of arithmetic operations required to factor the matrix is quite high. In this case, LOQO finds a good factorization for the reduced KKT matrix with 55803 nonzeros, which is quite sparse and only requires about 398,924 operations to factor. It takes 8.32 seconds to solve this problem. KNITRO, on the other hand, takes up to 552 conjugate gradient iterations per major iteration and takes 27.16 seconds. SNOPT struggles, performing 8773 iterations for a problem with 2240 degress of freedom, and takes 1685.39 seconds.

(4) *Unconstrained QP*: The performance of LOQO depends very much on the density of the Hessian for these problems. The three problems in this category are sparse, and LOQO easily outperforms the other solvers, as can be seen in the results from Table 5.

- *tridia*: This problem has 10000 free variables. The solution is $x_j = 0.5^{j-1}$. However, because of free variable splitting in LOQO, there are slacks in the algorithm that get initialized at the default value of 1 and must then be brought to their optimal values iteratively. LOQO solves this problem in 12 iterations and 2.09 seconds. Even if the optimal solution is provided to the model as the initial solution, it still takes 8 iterations and 1.73 seconds to find the corresponding optimal values for the slack variables.

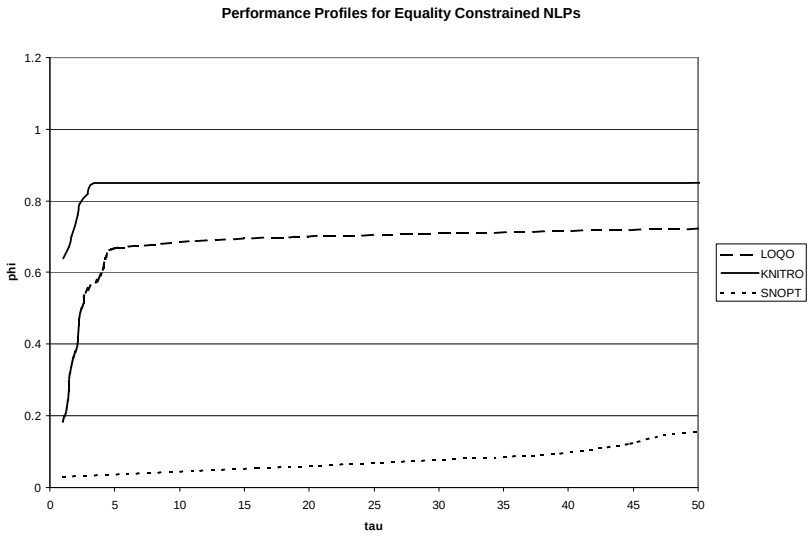

FIGURE 3. Performance Profiles for Equality Constrained NLPs.

KNITRO solves the problem with the default initial value in 5.72 seconds and SNOPT ran for several hours without making any progress, due to the many degrees of freedom. When the optimal solution is provided as the initial value, KNITRO exits with the error message that the step size is smaller than the machine precision, but SNOPT detects that the solution is optimal in 0 iterations and 0.40 seconds.

(5) *Equality Constrained NLP*: The performance profiles for the solvers are given by Figure 3. KNITRO outperforms LOQO for this set of problems, and SNOPT reaches its time limit due to the effort required by the limited-memory quasi-Newton approach.

- *gilbert*: This problem has one constraint that has quadratic terms for each variable. Even though the Hessian is diagonal, the one dense row in the reduced KKT matrix fills up the Cholesky factor when using the default settings in LOQO. The resulting factor has 500500 nonzeros, and LOQO takes 362.62 seconds to solve this problem. However, when

the "primal ordering" is requested, the nonzeros of the Jacobian cannot diffuse into the rest of the reduced KKT matrix, so we have a problem with only 1000 nonzeros and LOQO solves it in 0.73 seconds. This runtime is comparable to KNITRO's runtime of 0.72 seconds. SNOPT solves this problem in 1351.52 seconds.

- *dtoc4*: This problem has an objective function with 5000 terms and 4999 nonlinear equality constraints. Therefore, the evaluation times for the functions and the Hessian take up about half of LOQO's runtime of 13.00 seconds. KNITRO solves this problem in 3.05 seconds, and SNOPT times out.

(6) *Inequality Constrained NLP*: As can be seen in Table 9, LOQO outperforms KNITRO and SNOPT on this set of problems.
   - *cops/polygon*: LOQO performs 364 iterations, most of them with perturbations to the reduced KKT matrix and filter cuts, so it takes 38.32 seconds to solve this problem. However, if a nondefault initial solution for the slack variables is given to be 0.01, LOQO then solves this problem in 37 iterations and 2.72 seconds.
     KNITRO takes 7.99 seconds on this problem, and SNOPT solves it in 17.34 seconds.
   - *svanberg*: LOQO solves this problem in 13.19 seconds, but KNITRO performs up to 1260 conjugate gradient iterations per major iteration and takes 154.82 seconds. SNOPT times out on this problem.

(7) *NLPs with both Equality and Inequality Constraints*: On this set of problems, KNITRO outperforms the other solvers largely due to the fact that it solves most of these problems on the defaults. LOQO struggles on several problem because of the initialization of the slack variables on the equality constraints, but, given proper tuning, it solves most of these problems. The performance profile for this category of problems is given in Figure 4.
   - *cops/gasoil*: This problem has 20001 variables and 19998 constraints. The Jacobian of the constraints, A, is quite large and has 175974 nonzeros. LOQO solves the problem in 186.41 seconds. However, to compute the vertical step given by (11), KNITRO needs to first factor $\hat{A}^T\hat{A}$, where $\hat{A}$ is defined by (10). Due to the size of $\hat{A}$ and the density of $\hat{A}^T\hat{A}$, this computation requires a considerable amount of time per iteration. KNITRO solves the problem in 1158.95

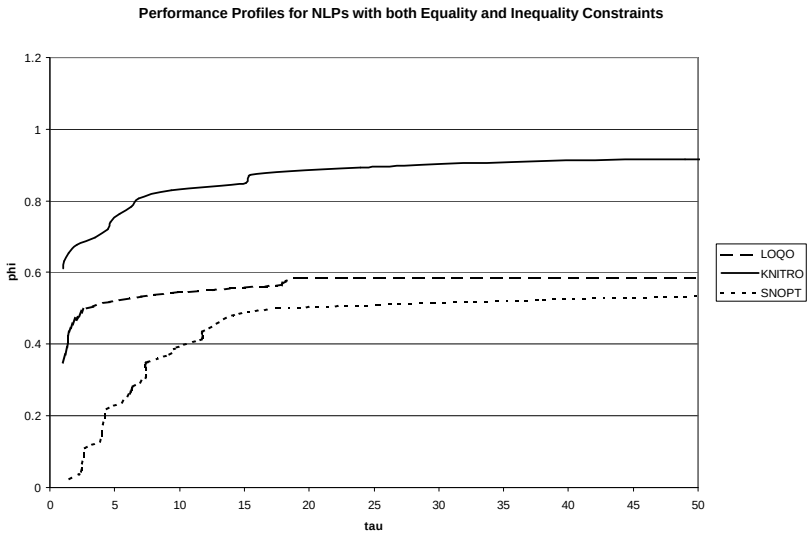

FIGURE 4. Performance Profiles for NLPs with both Equality and Inequality Constraints.

seconds. SNOPT faces the same problem and takes 3277.85 seconds.

- *braess/trafequilsf*: This problem has a full Hessian, and LOQO suffers because of the density of the Cholesky factor. It takes 67.50 seconds to solve the problem. KNITRO only takes 3.55 seconds, and SNOPT solves the problem in 21.89 seconds.
- *clnlbeam*: LOQO needs 105 iterations, most of which have perturbations to the Hessian, to solve this problem. It takes 6.42 seconds to solve it. KNITRO performs up to 7 conjugate gradient iterations per major iteration and solves this problem in 0.68 seconds. SNOPT takes 5.06 seconds.
- *dallasl*: LOQO solves this problem in 1.76 seconds. KNITRO, on the other hand, performs many major iterations, with up to 239 conjugate gradient iterations each, so it takes 27.16 seconds. SNOPT runs into an error while evaluating the nonlinear functions and quits.

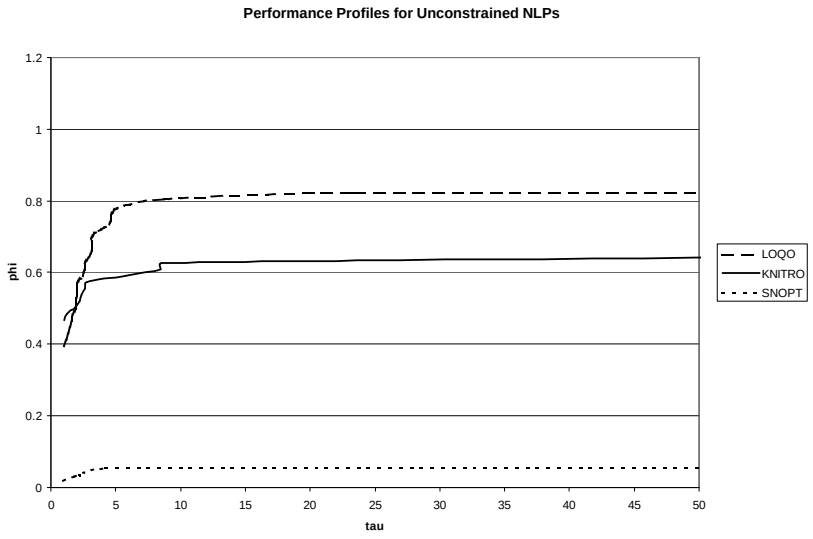

FIGURE 5. Performance Profiles for Unconstrained NLPs.

(8) *Unconstrained NLP*: This category is similar to its QP counterpart, in that sparsity is quite important. A performance profile for the solvers is provided in Figure 5.
   - *curly10*: This problem has a sparse Hessian with 209890 nonzeros. LOQO finds a good ordering for the reduced KKT matrix and solves this problem in 15.35 seconds. KNITRO, on the other hand, performs up to 20001 conjugate gradient iterations per major iteration and as a result requires 954.58 seconds. SNOPT times out on this problem as it has 10000 degrees of freedom.
   - *penalty1*: This problem has a fourth-order polynomial in the objective function. The size of the problem is relatively small, but the Hessian is full, so LOQO is quite slow with the factorization of the reduced KKT matrix given by (7). The runtime of LOQO is 344.40 seconds, as opposed to KNITRO which takes 14.70 seconds to solve the problem. SNOPT solves this problem in 61.84 seconds.

7. Conclusion.

As shown in the numerical results section, each of the algorithms presented and tested in this study has certain classes of problems on which they work well and others on which they perform poorly. Here are some of the results we found after numerical testing:

- Results on sparse NLPs with sparse factorizations for the reduced KKT matrix strongly indicate that a true Newton step is more efficient than one employing conjugate gradients on this type of problem.
- For dense factorizations, a conjugate gradient approach to solving the Newton equations can be efficient if the matrix is sufficiently well-conditioned.
- For a QP with all linear equality constraints, problems should be solved directly and not converted to an inequality constrained problem for use with an interior-point method as LOQO currently does. A QP with all inequality constraints, on the other hand, is solved best using an interior-point method such as LOQO. However, when the constraints are mixed (both equalities and inequalities), it remains to be determined whether LOQO, KNITRO, or an alternative approach will prove most successful in identifying the active set of the inequality constraints and solving them most efficiently. Computational work we have performed for this study is inconclusive on the best way to solve this type of problem.
- For constrained problems, LOQO's splitting of the free variables has proved quite effective in improving the conditioning of the reduced KKT matrix [10]. For unconstrained problems, adding slacks for which initial estimates are unknown can complicate the problem unnecessarily.
- SNOPT works on a reduced-space of the variables by using the constraints, and therefore, it can efficiently solve problems with small degrees of freedom. Because it uses only first-order information to estimate the reduced Hessian, by using a limited memory BFGS, the results clearly show that when the degree of freedom is large, a quasi-Newton method is not competitive with a Newton approach.
- For NLPs with equality constraints, the default initialization parameters for LOQO often fail, as indicated by the number of problems listed in Tables 8, 9, 11, 12 as tuned. These are problems solved succesfully with different initial parameters.

This indicates that further research on good default setting is necessary.

As we gain more experience in building and testing codes on a wide-variety of models, other issues might arise. So far, the results on the efficiency of state-of-the-art algorithms are quite encouraging.

References

[1] H.Y. Benson, D.F. Shanno, and R.J. Vanderbei. Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions. Technical Report ORFE 00-06, Department of Operations Research and Financial Engineering, Princeton University, 2000.

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

[3] A.R. Conn, N. Gould, and Ph.L. Toint. Constrained and unconstrained testing environment. http://www.dci.clrc.ac.uk/Activity.asp?CUTE.

[4] A.R. Conn, N.I.M. Gould, and Ph.L. Toint. *LANCELOT: a Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Verlag, Heidelberg, New York, 1992.

[5] E. D. Dolan and J. J. Moré. Benchmarking optimization software with COPS. Technical Report ANL/MCS-246, Argonne National Laboratory, November 2000.

[6] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201–214, 2002.

[7] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. Technical Report NA/171, University of Dundee, Dept. of Mathematics, Dundee, Scotland, 1997.

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

[9] P.E. Gill, W. Murray, and M.A. Saunders. User's guide for SNOPT 5.3: A Fortran package for large-scale nonlinear programming. Technical report, Systems Optimization Laboratory, Stanford University, Stanford, CA, 1997.

[10] D.F. Shanno and R.J. Vanderbei. Interior-point methods for nonconvex nonlinear programming: Orderings and higher-order methods. *Math. Prog.*, 87(2):303–316, 2000.

[11] R.J. Vanderbei. AMPL models. http://orfe.princeton.edu/r̃vdb/ampl/nlmodels.

[12] R.J. Vanderbei. A comparison between the minimum-local-fill and minimum-degree algorithms. Technical report, AT&T Bell Laboratories, 1990.

[13] R.J. Vanderbei. Symmetric quasi-definite matrices. *SIAM Journal on Optimization*, 5(1):100–113, 1995.

[14] R.J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 12:451–484, 1999.

[15] R.J. Vanderbei and D.F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–252, 1999.

Hande Y. Benson, Princeton University, Princeton, NJ

David F. Shanno, Rutgers University, New Brunswick, NJ

Robert J. Vanderbei, Princeton University, Princeton, NJ

Problem	n	m	LOQO	KNITRO	SNOPT
aug2d	20192	9996	27.47	5.36	(Time)
aug2dc	20200	9996	28.76	4.34	(Time)
aug3d	3873	1000	5.54	0.78	585.03
aug3dc	3873	1000	5.56	0.54	1657.08
dtoc3	14996	9997	23.83	2.99	597.32
gridnetb	13284	6724	14.10	4.30	(Time)
hager1	10000	5000	(Tuned)	0.97	(Time)
hager2	10000	5000	(Tuned)	1.54	(Time)
hager3	10000	5000	(Tuned)	1.85	(Time)

TABLE 2. Runtime results on equality constrained QPs. (Time) denotes that the algorithm timed out, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
biggsb1	1000	0	0.43	18.57	(IL)
chenhark	1000	0	0.27	473.70	(IL)
cvxbqp1	10000	0	40.88	4.96	121.86
ksip	20	1000	1.92	5.22	25.85
liswet1	10002	10000	(Tuned)	4.75	(ERROR)
liswet10	10002	10000	(Tuned)	5.87	(ERROR)
liswet11	10002	10000	(Tuned)	4.48	(ERROR)
liswet12	10002	10000	(Tuned)	5.17	(ERROR)
liswet2	10002	10000	(Tuned)	5.70	4.82
liswet3	10002	10000	10.02	5.43	11.11
liswet4	10002	10000	15.10	4.45	10.13
liswet5	10002	10000	11.71	4.37	12.77
liswet6	10002	10000	13.79	4.80	11.01
liswet7	10002	10000	(Tuned)	5.51	(ERROR)
liswet8	10002	10000	(Tuned)	5.83	(ERROR)
liswet9	10002	10000	(Tuned)	5.19	(ERROR)
mosarqp1	2500	700	1.04	13.60	344.92
mosarqp2	900	600	0.89	6.18	77.89
ncvxbqp1	10000	0	(Tuned)	17.46	121.96
ncvxbqp2	10000	0	(Tuned)	36.79	142.78
ncvxbqp3	10000	0	(Tuned)	85.45	175.30
oet1	3	1002	0.23	10.48	0.83
oet3	4	1002	0.28	5.91	0.70
pentdi	1000	0	0.48	0.37	0.04
powell20	1000	1000	0.53	1.23	2.38
sipow1	2	10000	4.84	8.55	55.34
sipow1m	2	10000	4.71	9.23	55.04
sipow2	2	5000	1.60	2.86	18.78
sipow2m	2	5000	1.61	3.00	18.75
sipow3	4	9998	5.85	14.28	6.33

TABLE 3. Runtime results on inequality constrained QPs. (IL) denotes that the algorithm reached its iteration limit, (ERROR) denotes that the algorithm exited with an error message, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
sipow4	4	10000	7.17	20.85	8.63
tfi2	3	10000	6.62	(ERROR)	8.26
yao	2000	1999	17.23	1.41	(ERROR)
cops/bearing	10000	0	21.18	240.83	(Time)
cops/torsion	2500	0	21.17	582.23	(Time)
nnls/nnls	1200	0	2068.59	1539.96	530.03

TABLE 4. Runtime results on inequality constrained QPs. (Time) denotes that the algorithm timed out, (ERROR) denotes that the algorithm exited with an error message.

Problem	n	m	LOQO	KNITRO	SNOPT
aug2dcqp	20200	9996	22.53	89.49	(Time)
aug2dqp	20192	9996	22.99	73.71	(Time)
aug3dcqp	3873	1000	2.48	22.49	869.33
aug3dqp	3873	1000	2.78	20.96	196.82
blockqp1	2005	1001	1.03	1.26	15.49
blockqp2	2005	1001	0.72	1.77	18.18
blockqp3	2005	1001	1.81	2.38	17.53
blockqp4	2005	1001	1.10	1.97	18.18
blockqp5	2005	1001	1.86	3.33	16.03
bloweya	2002	1002	4.70	1.01	2.79
bloweyb	2002	1002	8.37	1.18	1.98
bloweyc	2002	1002	1.68	0.63	2.01
cvxqp1	1000	500	7.03	1.73	6.03
cvxqp2	10000	2500	1609.03	288.39	(Time)
cvxqp3	10000	7500	6173.21	5.60	417.44
gouldqp2	699	349	0.45	2.45	7.23
gouldqp3	699	349	0.30	0.92	1.40
gridneta	8964	6724	8.32	27.16	1685.39
gridnetc	7564	3844	14.82	67.98	(Time)
hager4	10000	5000	7.80	8.05	(Time)
hues-mod	10000	2	41.59	(IL)	(Time)
huestis	10000	2	8.09	(IL)	(Time)
ncvxqp1	1000	500	357.97	2.17	1.63
ncvxqp2	1000	500	171.22	1.82	2.60
ncvxqp3	1000	500	56.48	3.09	4.96
ncvxqp4	1000	250	191.04	1.61	1.48
ncvxqp5	1000	250	50.23	1.70	1.63
ncvxqp6	1000	250	51.29	4.08	2.75
ncvxqp7	1000	750	456.35	1.81	1.98
ncvxqp8	1000	750	178.97	2.91	2.21

TABLE 5. Runtime results on QPs with both equality and inequality constraints. (Time) denotes that the algorithm timed out, (IL) denotes that the algorithm reached its iteration limit, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
ncvxqp9	1000	750	104.51	2.81	4.21
nnls/nnls2	1249	949	1.39	5.11	9.55
reading2	15001	10000	11.26	7.69	996.02
sosqp2	20000	10001	17.44	37.38	(Time)
ubh1	17997	12000	27.93	3211.41	(Time)
markowitz/markowitz2	1200	201	557.56	130.83	192.51
stengel/ex3.3.1a	1599	1598	0.61	0.27	6.68
stengel/ex3.3.1b	2399	2397	3.20	0.61	11.61
stengel/ex3.3.1c	2399	2397	1.44	0.62	11.54
structure/structure4	720	1537	17.86	(IL)	(Time)

TABLE 6. Runtime results on QPs with both equality and inequality constraints. (Time) denotes that the algorithm timed out, (IL) denotes that the algorithm reached its iteration limit, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
dqdrtic	5000	0	1.18	0.39	(Time)
power	1000	0	0.19	1.02	695.27
tridia	10000	0	2.09	5.72	(Time)

TABLE 7. Runtime results on unconstrained QPs. (Time) denotes that the algorithm timed out.

Problem	n	m	LOQO	KNITRO	SNOPT
artif	5000	5000	5.36	1.07	(Time)
bdvalue	5000	5000	1.54	0.26	0.09
bratu2d	4900	4900	7.61	10.93	(Time)
bratu2dt	4900	4900	(Tuned)	13.34	(Time)
bratu3d	3375	3375	62.76	213.14	(Time)
broydn3d	10000	10000	4.42	1.22	(Time)
broydnbd	5000	5000	5.95	2.16	(Time)
cbratu2d	882	882	0.74	0.59	245.30
cbratu3d	1024	1024	3.04	1.37	122.78
drcav1lq	10000	10000	(Time)	(ERROR)	(Time)
drcav2lq	10000	10000	(Time)	(ERROR)	(Time)
drcav3lq	10000	10000	(Time)	(ERROR)	(Time)
dtoc1l	14985	9990	11.27	6.16	(Time)
dtoc1na	1485	990	5.74	3.85	145.76
dtoc1nb	1485	990	4.84	3.43	153.71
dtoc1nc	1485	990	9.44	4.41	215.53
dtoc1nd	735	490	4.41	2.69	55.79
dtoc2	5994	3996	5.64	3.87	797.19
dtoc4	14996	9997	13.00	3.05	(Time)
dtoc5	9998	4999	6.20	1.52	(Time)
dtoc6	10000	5000	7.94	3.42	(Time)
gilbert	1000	1	362.62	0.72	1351.52
msqrta	1024	1024	252.67	571.51	(IL)
msqrtb	1024	1024	277.90	531.71	(IL)
orthrdm2	4003	2000	104.68	1.01	3363.35
orthregc	10005	5000	29.42	11.48	(Time)
orthregd	10003	5000	(Tuned)	(Time)	(Time)
orthrgdm	10003	5000	399.49	6.66	(Time)
orthrgds	10003	5000	(Tuned)	(Time)	(Time)
porous1	4900	4900	14.90	31.47	(Time)

TABLE 8. Runtime results on equality constrained NLPs. (Time) denotes that the algorithm timed out, (IL) denotes that the algorithm reached its iteration limit, (ERROR) denotes that the algorithm exited with an error message, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
porous2	4900	4900	12.53	20.10	(Time)
rocket/moonshot2	5988	3994	4.55	2.19	2191.16
rocket/moonshot3	3992	1998	226.62	1.59	1959.94

TABLE 9. Runtime results on equality constrained NLPs. (Time) denotes that the algorithm timed out.

Problem	n	m	LOQO	KNITRO	SNOPT
cresc132	6	2654	(IL)	(Time)	762.96
manne	1094	730	(IL)	3.49	1.77
mccormck	50000	0	18.43	(ERROR)	(Time)
ngone	97	1273	6.23	11.95	2.33
nonscomp	10000	0	8.80	(IL)	(Time)
oet2	3	1002	3.09	25.64	2.05
oet7	7	1002	(Tuned)	26.64	60.68
scon1dls	1000	0	15.66	(IL)	(IL)
sinrosnb	1000	999	0.20	(ERROR)	0.05
svanberg	5000	5000	13.19	154.82	(Time)
cops/minsurf	10000	0	8.87	(IL)	(Time)
cops/polygon	98	1273	38.32	7.99	9.31
facloc/esfl_socp	5002	5000	4.64	4.74	6.58
springs/springs	2998	1000	33.47	(IL)	(Time)
structure/structure3	640	4216	9.86	(ERROR)	914.12

TABLE 10. Runtime results on inequality constrained NLPs. (Time) denotes that the algorithm timed out, (IL) denotes that the algorithm reached its iteration limit, (ERROR) denotes that the algorithm exited with an error message, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
bigbank	1773	814	1.31	64.10	(Time)
brainpc0	6905	6900	(Tuned)	51.14	378.32
brainpc1	6905	6900	(Tuned)	194.94	797.75
brainpc2	13805	13800	(Tuned)	329.19	793.75
brainpc3	6905	6900	(Tuned)	71.00	836.21
brainpc4	6905	6900	(Tuned)	64.89	836.21
brainpc5	6905	6900	278.39	193.01	820.16
brainpc6	6905	6900	(Tuned)	199.29	795.96
brainpc7	6905	6900	180.02	367.99	801.20
brainpc8	6905	6900	(Tuned)	68.69	804.86
brainpc9	6905	6900	(Tuned)	333.45	826.44
clnlbeam	1499	1000	6.42	0.68	5.06
corkscrw	8997	7000	14.22	(IL)	1611.34
dallasl	837	598	1.76	27.16	(ERROR)
gausselm	1495	3690	(IL)	23.18	(ERROR)
gridnetd	3945	2644	7.48	10.34	48.20
gridnete	7565	3844	10.30	5.77	(Time)
gridnetf	7565	3844	22.72	121.22	(Time)
kissing	127	903	(Tuned)	111.04	(Inf)
optcdeg2	1198	799	2.01	1.41	13.89
optcdeg3	1198	799	1.67	1.43	13.20
reading1	10001	5000	(Tuned)	31.15	(Time)
sawpath	589	782	2.82	26.30	4.35
semicon1	1000	1000	(Tuned)	4.56	(IL)
semicon2	1000	1000	1.94	0.68	(IL)
sreadin3	10000	5000	14.55	2.70	(Time)
trainf	20000	10002	59.21	891.67	(Time)
trainh	20000	10002	(Tuned)	2011.92	(Time)
ubh5	19997	14000	(Tuned)	(IL)	(Time)
cops/gasoil	20001	19998	186.41	1158.95	3277.85

TABLE 11. Runtime results on NLPs with both equality and inequality constraints. (Time) denotes that the algorithm timed out, (Inf) denotes that the SNOPT concluded that the problem is infeasible, (IL) denotes that the algorithm reached its iteration limit, (ERROR) denotes that the algorithm exited with an error message, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
cops/methanol	12002	11997	247.95	186.90	2640.54
cops/pinene	10000	9995	22.96	10.25	1070.19
cops/robot	8999	6001	(Tuned)	22.70	(Time)
cops/rocket	16001	12000	105.35	107.20	(Time)
cops/steering	5000	4001	(Tuned)	6.29	1205.29
brachistochrone/bc4	16383	16384	27.72	95.78	(Time)
braess/trafequil	722	361	40.40	2.26	8.66
braess/trafequil2	798	437	0.52	2.30	1.40
braess/trafequil2sf	932	385	0.59	2.72	3.28
braess/trafequilsf	856	309	67.50	3.55	21.89
facloc/emfl_socp	5350	5300	(IL)	210.40	574.62
facloc/emfl_socp_vareps	5351	5300	8.50	(IL)	507.50
hang/hang_midpt	1499	2000	(IL)	(IL)	(Time)
minsurf/minsurf_socp	3009	2046	2.75	65.85	(Time)
stengel/ex3.5.1	1179	888	(IL)	36.25	1378.45
wbv/antenna2	24	1197	3.97	27.02	29.42

TABLE 12. Runtime results on NLPs with both equality and inequality constraints. (Time) denotes that the algorithm timed out, (IL) denotes that the algorithm reached its iteration limit, (ERROR) denotes that the algorithm exited with an error message, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
arwhead	5000	0	1.67	0.37	(Time)
bdexp	5000	0	4.17	0.87	(Time)
bdqrtic	1000	0	0.47	0.35	124.87
bratu1d	1001	0	(Tuned)	1.31	(IL)
broydn7d	1000	0	(IL)	2.18	1198.03
brybnd	5000	0	8.44	4.14	(Time)
chainwoo	1000	0	1.67	(IL)	(IL)
clplatea	4970	0	2.61	2.23	(Time)
clplateb	4970	0	3.61	8.38	(Time)
clplatec	4970	0	2.80	16.05	(Time)
cosine	10000	0	3.85	(ERROR)	(Time)
cragglvy	5000	0	2.83	(ERROR)	(Time)
curly10	10000	0	15.35	954.58	(Time)
curly20	10000	0	29.31	(Time)	(Time)
curly30	10000	0	47.16	(Time)	(Time)
dixmaana	3000	0	0.90	0.35	1522.23
dixmaanb	3000	0	1.98	0.81	1613.16
dixmaanc	3000	0	2.22	0.70	1669.69
dixmaand	3000	0	2.29	0.77	1672.60
dixmaane	3000	0	1.55	2.81	2667.47
dixmaanf	3000	0	4.12	2.11	2656.85
dixmaang	3000	0	3.51	2.08	2711.34
dixmaanh	3000	0	4.13	2.13	2396.91
dixmaani	3000	0	1.63	13.68	(Time)
dixmaanj	3000	0	4.06	11.06	2526.37
dixmaank	3000	0	4.44	11.50	(Time)
dixmaanl	3000	0	4.52	9.85	3521.97
dqrtic	5000	0	5.20	1.39	(Time)
drcavty1	10000	0	(Time)	(IL)	(Time)
drcavty2	10000	0	(Time)	(IL)	(Time)

TABLE 13. Runtime results on unconstrained NLPs. (Time) denotes that the algorithm timed out, (IL) denotes that the algorithm reached its iteration limit, (ERROR) denotes that the algorithm exited with an error message, (Tuned) denotes that LOQO can solve this problem with tuning.

Problem	n	m	LOQO	KNITRO	SNOPT
drcavty3	10000	0	(Time)	(Time)	(Time)
edensch	2000	0	0.59	0.37	480.78
engval1	5000	0	1.88	(ERROR)	(Time)
fletcbv3	10000	0	(ERROR)	(IL)	(Time)
fletchbv	10000	0	(IL)	(IL)	(Time)
fminsrf2	1024	0	(Tuned)	2.47	(Time)
fminsurf	1024	0	364.16	116.48	250.41
freuroth	5000	0	3.15	(ERROR)	(Time)
indef	1000	0	(ERROR)	(IL)	(Unb)
liarwhd	10000	0	8.10	2.55	(Time)
lminsurf	15129	0	346.25	(IL)	(Time)
morebv	5000	0	1.23	19.85	0.14
msqrtals	1024	0	855.79	575.24	(IL)
msqrtbls	1024	0	778.64	382.13	(IL)
nondia	9999	0	4.63	0.82	(Time)
nondquar	10000	0	6.74	57.37	(Time)
penalty1	1000	0	344.40	14.70	61.84
quartc	10000	0	12.47	2.69	(Time)
scosine	10000	0	27.13	(IL)	(Time)
scurly10	10000	0	109.52	(Time)	(Time)
scurly20	10000	0	182.77	(Time)	(Time)
scurly30	10000	0	273.64	(Time)	(Time)
sinquad	10000	0	59.38	72.55	(Time)
srosenbr	10000	0	5.01	1.91	(Time)
woods	10000	0	12.17	6.03	(Time)
brachistochrone/bc7	16383	0	(IL)	(ERROR)	(Time)

TABLE 14. Runtime results on unconstrained NLPs. (Time) denotes that the algorithm timed out, (IL) denotes that the algorithm reached its iteration limit, (Unb) denotes that SNOPT reported that the problem is unbounded, (ERROR) denotes that the algorithm exited with an error message, (Tuned) denotes that LOQO can solve this problem with tuning.