A Numerical Study of Active-Set and Interior-Point Methods for Bound Constrained Optimization *

Long Hei[1], Jorge Nocedal[2], Richard A. Waltz[2]

[1] Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston IL 60208, USA.
[2] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston IL 60208, USA.

Summary. This papers studies the performance of several interior-point and active-set methods on bound constrained optimization problems. The numerical tests show that the sequential linear-quadratic programming (SLQP) method is robust, but is not as effective as gradient projection at identifying the optimal active set. Interior-point methods are robust and require a small number of iterations and function evaluations to converge. An analysis of computing times reveals that it is essential to develop improved preconditioners for the conjugate gradient iterations used in SLQP and interior-point methods. The paper discusses how to efficiently implement incomplete Cholesky preconditioners and how to eliminate ill-conditioning caused by the barrier approach. The paper concludes with an evaluation of methods that use quasi-Newton approximations to the Hessian of the Lagrangian.

1 Introduction

A variety of interior-point and active-set methods for nonlinear optimization have been developed in the last decade; see Gould et al. [12] for a recent survey. Some of these algorithms have now been implemented in high quality software and complement an already rich collection of established methods for constrained optimization. It is therefore an appropriate time to evaluate the contributions of these new algorithms in order to identify promising directions of future research. A comparison of active-set and interior-point approaches is particularly interesting given that both classes of algorithms have matured.

A practical evaluation of optimization algorithms is complicated by details of implementation, heuristics and algorithmic options. It is also difficult

* This work was supported by National Science Foundation grant CCR-0219438, Department of Energy grant DE-FG02-87ER25047-A004 and a grant from the Intel Corporation.

to select a good test set because various problem characteristics, such as non-convexity, degeneracy and ill-conditioning, affect algorithms in different ways. To simplify our task, we focus on large-scale bound constrained problems of the form

$$\underset{x}{\text{minimize}} \quad f(x) \tag{1a}$$

$$\text{subject to} \quad l \leq x \leq u, \tag{1b}$$

where $f : \mathrm{R}^n \to \mathrm{R}$ is a smooth function and $l \leq u$ are both vectors in R^n. The simple geometry of the feasible region (1b) eliminates the difficulties caused by degenerate constraints and allows us to focus on other challenges, such as the effects of ill-conditioning.

Furthermore, the availability of specialized (and very efficient) gradient projection algorithms for bound constrained problems places great demands on the general-purpose methods studied in this paper. The gradient projection method can quickly generate a good working set and then perform subspace minimization on a smaller dimensional subspace. Interior-point methods, on the other hand, never eliminate inequalities and work on an n-dimensional space, putting them at a disadvantage (in this respect) when solving bound constrained problems.

We chose four active-set methods that are representative of the best methods currently available:

(1) The sequential quadratic programming (SQP) method implemented in SNOPT [10];
(2) The sequential linear-quadratic programming (SLQP) method implemented in KNITRO/ACTIVE [2];
(3) The gradient projection method implemented in TRON [15];
(4) The gradient projection method implemented in L-BFGS-B [4, 19].

SQP and gradient projection methods have been studied extensively since the 1980s, while SLQP methods have emerged in the last few years. These three methods are quite different in nature. The SLQP and gradient projection methods follow a so-called EQP approach in which the active-set identification and optimization computations are performed in two separate stages. In the SLQP method a linear program is used in the active-set identification phase, while the gradient projection performs a piecewise linear search along the gradient projection path. In contrast, SQP methods follow an IQP approach in which the new iterate and the new estimate of the active set are computed simultaneously by solving an inequality constrained subproblem.

We selected two interior-point methods, both of which are implemented in the KNITRO software package [5]:

(5) The primal-dual method in KNITRO/DIRECT [18] that (typically) computes steps by performing a factorization of the primal-dual system;
(6) The trust region method in KNITRO/CG [3] that employs iterative linear algebra techniques in the step computation.

The algorithm implemented in KNITRO/DIRECT is representative of various line search primal-dual interior-point methods developed since the mid 1990s (see [12]), whereas the algorithm in KNITRO/CG follows a trust region approach that is significantly different from most interior-point methods proposed in the literature. We have chosen the two interior-point methods available in the KNITRO package, as opposed to other interior-point codes, to minimize the effect of implementation details. In this way, the same type of stop tests and scalings are used in the two interior-point methods and in the SLQP method used in our tests.

The algorithms implemented in (2), (3) and (6) use a form of the conjugate gradient method in the step computation. We study these iterative approaches, giving particular attention to their performance in interior-point methods where preconditioning is more challenging [8, 1, 13]. Indeed, whereas in active-set methods ill-conditioning is caused only by the objective function and constraints, in interior-point methods there is an additional source of ill-conditioning caused by the barrier approach.

The paper is organized as follows. In Section 2, we describe numerical tests with algorithms that use exact Hessian information. The observations made from these results set the stage for the rest of the paper. In Section 3 we describe the projected conjugate gradient method that plays a central role in several of the methods studied in our experiments. A brief discussion on preconditioning for the SLQP method is given in Section 4. Preconditioning in the context of interior-point methods is the subject of Section 5. In Section 6 we study the performance of algorithms that use quasi-Newton Hessian approximations.

2 Some Comparative Tests

In this section we report test results for four algorithms, all using exact second derivative information. The algorithms are: TRON (version 1.2), KNITRO/DIRECT, KNITRO/CG and KNITRO/ACTIVE (versions 5.0). The latter three were not specialized in any way to the bound constrained case. In fact, we know of no such specialization for interior-point methods, although advantage can be taken at the linear algebra level, as we discuss below. A modification of the SLQP approach that may prove to be effective for bound constraints is investigated by Byrd and Waltz [6], but was not used here.

We do not include SNOPT in these tests because this algorithm works more effectively with quasi-Newton Hessian approximations, which are studied in Section 6. Similarly, L-BFGS-B is a limited memory quasi-Newton method and will also be discussed in that section. All the test problems were taken from the CUTEr collection [11] using versions of the models formulated in Ampl [9]. We chose all the bound constrained CUTEr problems available as Ampl models for which the sizes could be made large enough for our purposes,

while excluding some of the repeated models (e.g., we only used `torsion1` and `torsiona` from the group of torsion models).

The results are summarized in Table 1, which reports the number of variables for each problem, as well as the number of iterations, function evaluations and computing time for each solver. For TRON we also report the number of active bounds at the solution; for those solvers that use a conjugate gradient (CG) iteration, we report the average number of CG iterations per outer iteration. In addition, for KNITRO/CG we report the number of CG iterations performed in the last iteration of the optimization algorithm divided by the number of variables (`endCG/`n).

We use a limit of 10000 iterations for all solvers. Unless otherwise noted, default settings were used for all solvers, including default stopping tests and tolerances which appeared to provide comparable solution accuracy in practice.

We also provide in Figures 1 and 2 performance profiles based, respectively, on the number of function evaluations and computing time. All figures plot the logarithmic performance profiles described in [7].

		TRON					KNITRO/DIRECT			KNITRO/CG					KNITRO/ACTIVE			
problem	n	iter	feval	CPU	actv@sol	aveCG	iter	feval	CPU	iter	feval	CPU	aveCG	endCG/n	iter	feval	CPU	aveCG
biggsb1	20000	X_1	X_1	X_1	X_1	X_1	12	13	2.61	12	13	245.48	942.50	0.1046	X_1	X_1	X_1	X_1
bqpgauss	2003	206	206	6.00	95	6.93	20	21	0.88	42	43	183.85	3261.14	2.0005	232	234	65.21	1020.46
chenhark	20000	72	72	4.30	19659	1.00	18	19	2.57	20	21	1187.49	4837.60	0.7852	847	848	1511.60	1148.74
clnlbeam	20000	6	6	0.50	9999	0.83	11	12	2.20	12	13	2.60	3.67	0.0001	3	4	0.41	1.00
cvxbqp1	20000	2	2	0.11	20000	0.00	9	10	51.08	9	10	3.60	6.33	0.0003	1	2	0.18	0.00
explin	24000	8	8	0.13	23995	0.88	24	25	6.79	26	27	16.93	16.46	0.0006	13	14	1.45	3.08
explin2	24000	6	6	0.10	23997	0.83	26	27	6.39	25	26	16.34	16.72	0.0005	12	13	1.26	2.17
expquad	24000	X_2	X_2	X_2	X_2	X_2	X_4	X_4	X_4	X_4	X_4	X_4	X_4	X_4	183	663	56.87	1.42
gridgena	26312	16	16	14.00	0	1.75	8	23	17.34	7	8	43.88	160.86	0.0074	6	8	9.37	77.71
harkerp2	2000	X_3	X_3	X_3	X_3	X_3	15	16	484.48	27	28	470.76	12.07	0.0010	7	8	119.70	0.86
jnlbrng1	21904	30	30	6.80	7080	1.33	15	16	6.80	18	19	163.62	632.33	0.1373	39	40	27.71	92.23
jnlbrnga	21904	30	30	6.60	7450	1.37	14	16	6.31	18	19	184.75	708.67	0.1608	35	36	30.05	122.03
mccormck	100000	6	7	2.60	1	1.00	9	10	11.60	12	13	20.89	4.17	0.0001	X_5	X_5	X_5	X_5
minsurfo	10000	10	10	2.10	2704	3.00	367	1313	139.76	X_1	X_1	X_1	X_1	X_1	8	10	4.32	162.33
ncvxbqp1	20000	2	2	0.11	20000	0.00	35	36	131.32	32	33	10.32	4.63	0.0006	3	4	0.36	0.67
ncvxbqp2	20000	8	8	0.50	19869	1.13	75	76	376.01	73	74	58.65	26.68	0.0195	30	39	5.90	4.26
nobndtor	32400	34	34	10.00	5148	2.85	15	16	8.66	13	14	6817.52	24536.62	2.0000	66	67	78.85	107.42
nonscomp	20000	8	8	0.82	0	0.88	21	23	5.07	129	182	81.64	12.60	0.0003	10	11	1.37	4.20
obstclae	21904	31	31	4.90	10598	1.84	17	18	7.66	17	18	351.83	846.00	0.3488	93	116	40.36	37.01
obstclbm	21904	25	25	4.20	5262	1.64	12	13	5.52	11	12	562.34	2111.64	0.1819	43	50	16.91	39.08
pentdi	20000	2	2	0.17	19998	0.50	12	13	2.24	14	15	3.40	5.36	0.0005	1	2	0.21	1.00
probpenl	5000	2	2	550.00	1	0.50	3	4	733.86	3	4	6.41	1.00	0.0002	1	2	2.79	1.00
qrtquad	5000	28	58	1.60	5	2.18	39	63	1.56	X_5	X_5	X_5	X_5	X_5	783	2403	48.44	2.02
qudlin	20000	2	2	0.02	20000	0.00	17	18	2.95	24	25	12.74	16.08	0.0004	3	4	0.20	0.67
reading1	20001	8	8	0.78	20001	0.88	16	17	6.11	14	15	5.64	7.21	0.0001	3	4	0.44	0.33
scond1ls	2000	592	1748	18.00	0	2.96	1276	4933	754.57	1972	2849	10658.26	2928.17	0.3405	X_1	X_1	X_1	X_1
sineali	20000	11	15	1.30	0	1.27	9	12	3.48	18	61	13.06	4.57	0.0001	34	112	8.12	1.58
torsion1	32400	59	59	14.00	9824	1.86	11	12	8.13	7	8	359.78	1367.14	0.2273	65	66	57.53	64.65
torsiona	32400	59	59	16.00	9632	1.88	10	11	7.62	6	7	80.17	348.33	0.0279	62	63	62.43	74.06

X_1: iteration limit reached
X_2: numerical result out of range
X_3: solver did not terminate
X_4: current solution estimate cannot be improved
X_5: relative change in solution estimate $< 10^{-15}$

Table 1. Comparative results of four methods that use exact second derivative information

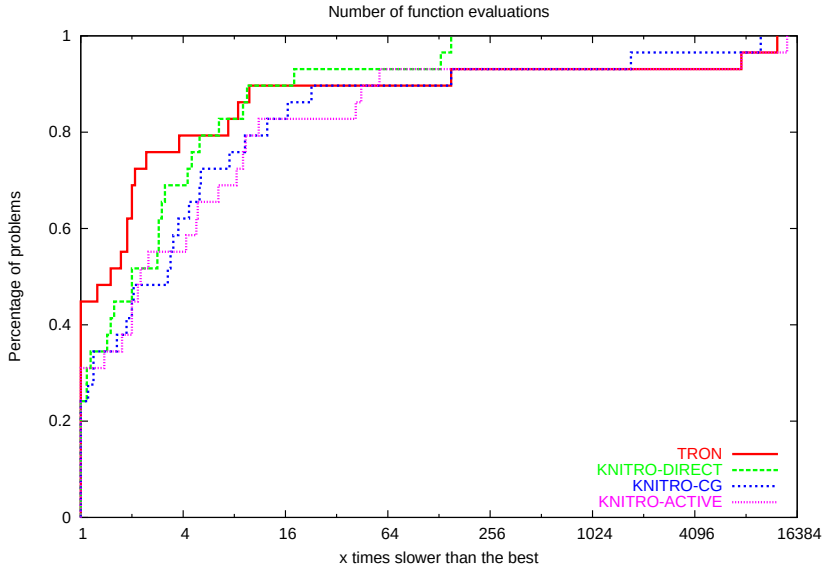

Fig. 1. Number of Function Evaluations

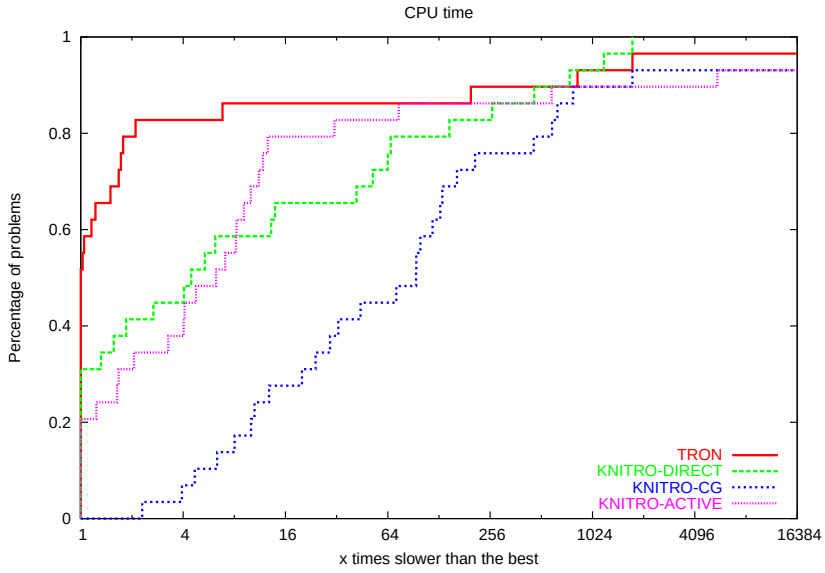

Fig. 2. CPU Time

We now comment on these results.

In terms of robustness, there appears to be no significant difference between the four algorithms tested, although KNITRO/DIRECT is slightly more reliable.

Function Evaluations. In terms of function evaluations (or iterations), we observe some significant differences between the algorithms. KNITRO/ACTIVE requires more iterations overall than the other three methods; if we compare it with TRON—the other active-set method—we note that TRON is almost uniformly superior. This suggests that the SLQP approach implemented in KNITRO/ACTIVE is less effective than gradient projection at identifying the optimal active set. We discuss this issue in more detail below.

As expected, the interior-point methods typically perform between 10 and 30 iterations to reach convergence. Since the geometry of bound constraints is simple, only nonlinearity and nonconvexity in the objective function cause interior-point methods to perform a large number of iterations. It is not surprising that KNITRO/CG requires more iterations than KNITRO/DIRECT given that it uses an inexact iterative approach in the step computation.

Figure 1 indicates that the gradient projection method is only slightly more efficient than interior-point methods, in terms of function evaluations. As in any active-set method, TRON sometimes converges in a very small number of iterations (e.g. 2), but on other problems it requires significantly more iterations than the interior-point algorithms.

CPU Time. It is clear from Table 1 that KNITRO/CG requires the largest amount of computing time among all the solvers. This test set contains a significant number of problems with ill-conditioned Hessians, $\nabla^2 f(x)$, and the step computation of KNITRO/CG is dominated by the large number of CG steps performed. TRON reports the lowest computing times; the average number of CG iterations per step is rarely greater than 2. This method uses an incomplete Cholesky preconditioner [14], whose effectiveness is crucial to the success of TRON.

The high number of CG iterations in KNITRO/CG is easily explained by the fact that it does not employ a preconditioner to remove ill-conditioning caused by the Hessian of the objective function. What is not so simple to explain is the higher number of CG iteration in KNITRO/CG compared to KNITRO/ACTIVE. Both methods use an unpreconditioned projected CG method in the step computation (see Section 3), and therefore one would expect that both methods would suffer equally from ill-conditioning. Table 1 indicates that this is not the case. In addition, we note that the average cost of the CG iteration is higher in KNITRO/CG than in KNITRO/ACTIVE.

One possible reason for this difference is that the SLQP method applies CG to a smaller problem than the interior-point algorithm. The effective number of variables in the KNITRO/ACTIVE CG iteration is $n - t_k$, where t_k is the number of constraints in the working set at the kth iteration. On the other hand, the interior-point approach applies the CG iteration in n-dimensional

space. This, however, accounts only partly for the differences in performance. For example, we examined some runs in which t_k is about $n/3$ to $n/2$ during the run of KNITRO/ACTIVE and noticed that the differences in CG iterations between KNITRO/CG and KNITRO/ACTIVE are significantly greater than 2 or 3 toward the end of the run. As we discuss in Section 5, it is the combination of barrier and Hessian ill-conditioning that can be very detrimental to the interior-point method implemented in KNITRO/CG.

Active-set identification. The results in Table 1 suggest that the SLQP approach will not be competitive with gradient projection on bound constrained problems, unless the SLQP method can be redesigned so as to require fewer outer iterations. In other words, it needs to improve its active-set identification mechanism. As already noted, the SLQP method in KNITRO/ACTIVE computes the step in two phases. In the linear programming phase, an estimate of the optimal active set is computed. This linear program takes a simple form in the bound constrained case, and can be solved very quickly. Most of the computing effort goes in the EQP phase, which solves an equality constrained quadratic program where the constraints in the working set are imposed as equalities (i.e., fixed variables in this case) and all other constraints are ignored. This subproblem is solved using a projected CG iteration. Assuming that the cost of this CG phase is comparable in TRON and KNITRO/ACTIVE (we can use the same preconditioners in the two methods), the SLQP method needs to perform a similar number of outer iterations to be competitive.

Comparing the detailed results of TRON versus KNITRO/ACTIVE highlights two features that provide TRON with superior active-set identification properties. First, the active set determined by SLQP is given by the solution of one LP (whose solution is constrained by an infinity norm trust-region), whereas the gradient projection method, minimizes a quadratic model along the gradient projection path to determine an active-set estimate. Because it explores a whole path as opposed to a single point, this often results in a better active-set estimate for gradient projection. An enhancement to SLQP proposed in [6] mimics what is done in gradient projection by solving a *parameteric* LP (parameterized by the trust-region radius) rather than a single LP to determine an active set with improved results.

Second, the gradient projection implementation in TRON has a feature which allows it to add bounds to the active set during the unconstrained minimization phase, if inactive bounds are encountered. On some problems this significantly decreases the number of iterations required to identify the optimal active set. In the bound constrained case, it is easy to do something similar for SLQP. In [6], this feature was added to an SLQP algorithm and shown to improve performance on bound constrained problems.

The combination of these two features may result in an SLQP method that is competitive with TRON. However, more research is needed to determine if this goal can be achieved.

In the following section we give attention to the issue of preconditioning. Although, in this paper, we are interested in preconditioners applied to bound constrained problems, we will first present our preconditioning approach in the more general context of constrained optimization where it is also applicable.

3 The Projected Conjugate Gradient Method

Both KNITRO/CG and KNITRO/ACTIVE use a projected CG iteration in the step computation. To understand the challenges of preconditioning this iteration, we now describe it in some detail.

The projected CG iteration is a method for solving equality constrained quadratic programs of the form

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}x^T Gx + h^T x \tag{2a}$$

$$\text{subject to} \quad Ax = b, \tag{2b}$$

where G is an $n \times n$ symmetric matrix that is positive definite on the null space of the $m \times n$ matrix A, and h is an n-vector. Problem (2) can be solved by eliminating the constraints (2b), applying the conjugate gradient method to the reduced problem of dimension $(n - m)$, and expressing this solution process in n-dimensional space. This procedure is specified in the following algorithm. We denote the preconditioning operator by P; its precise definition is given below.

Algorithm PCG. Preconditioned Projected CG Method.
Choose an initial point x_0 satisfying $Ax_0 = b$. Set $x \leftarrow x_0$, compute $r = Gx+h$, $z = Pr$ and $p = -z$.
Repeat the following steps, until $\|z\|$ is smaller than a given tolerance:

$$\begin{aligned} \alpha &= r^T z / p^T G p \\ x &\leftarrow x + \alpha p \\ r^+ &= r + \alpha G p \\ z^+ &= P r^+ \\ \beta &= (r^+)^T z^+ / r^T z \\ p &\leftarrow -z^+ + \beta p \\ z &\leftarrow z^+ \ \text{ and } \ r \leftarrow r^+ \end{aligned}$$

End

The preconditioning operation is defined indirectly, as follows. Given a vector r, we compute $z = Pr$ as the solution of the system

$$\begin{bmatrix} D & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \tag{3}$$

where D is a symmetric matrix that is required to be positive definite on the null space of A, and w is an auxiliary vector. A preconditioner of the form (3) is often called a *constraint preconditioner*. To accelerate the convergence of Algorithm PCG, the matrix D should approximate G in the null space of A and should be sparse so that solving (3) is not too costly. It is easy to verify that since initially $Ax_0 = b$, all subsequent iterates x of Algorithm PCG also satisfy the linear constraints (2b).

The choice $D = I$ gives an unpreconditioned projected CG iteration. To improve the performance of Algorithm PCG, we consider some other choices for D. One option is to let D be a diagonal matrix; see e.g. [1, 16]). Another option is to define D by means of an incomplete Cholesky factorization of G, but the challenge is how to implement it effectively in the setting of constrained optimization. An implementation that computes the incomplete factors L and L^T of G, multiplies them to give $D = LL^T$, and then factors the system (3), is of little interest; one might as well use the perfect preconditioner $D = G$. However, for special classes of problems, such as bound constrained optimization, it is possible to rearrange the computations and compute the incomplete Cholesky factorization on a reduced system, as discussed in the next sections.

We note that the KNITRO/CG and KNITRO/ACTIVE algorithms actually solve quadratic programs of the form (2) subject to a trust region constraint $\|x\| \leq \Delta$; in addition, G may not always be positive definite on the null space of A. To deal with these two requirements, Algorithm PCG can be adapted by following Steihaug's approach: we terminate the iteration if the trust region is crossed or if negative curvature is encountered [17]. In this paper, we will ignore these additional features and consider preconditioning in the simpler context of Algorithm PCG.

4 Preconditioning the SLQP Method

In the SLQP method implemented in KNITRO/ACTIVE, the equality constraints (2b) are defined as the linearization of the problem constraints belonging to the working set. We have already mentioned that this working set is obtained by solving an auxiliary linear program. In the experiments reported in Table 1, we used $D = I$ in (3), i.e. the projected CG iteration in KNITRO/ACTIVE was not preconditioned. This explains the high number of CG iterations and computing time for many of the problems.

Let us therefore consider other choices for D. Diagonal preconditioners are straightforward to implement, but are often not very effective. A more attractive option is incomplete Cholesky preconditioning, which can be implemented as follows.

Suppose for the moment that we use the perfect preconditioner $D = G$ in (3). Since z satisfies $Az = 0$, we can write $z = Zu$, where Z is a basis matrix such that $AZ = 0$ and u is some vector of dimension $(n - m)$. Multiplying the

first block of equations in (3) by Z^T and recalling the condition $Az = 0$ we have that

$$Z^T GZu = Z^T r. \tag{4}$$

We now compute the incomplete Cholesky factorization of the reduced Hessian,

$$LL^T \approx Z^T GZ, \tag{5}$$

solve the system

$$LL^T \hat{u} = Z^T r, \tag{6}$$

and set $z = Z\hat{u}$. This defines the preconditioning step. Since for nonconvex problems $Z^T GZ$ may not be positive definite, we can apply a *modified* incomplete Cholesky factorization of the form $LL^T \approx Z^T(G + \delta I)Z$, for some positive scalar δ; see [14].

For bound constrained problems, the linear constraints (2*b*) are defined to be the bounds in the working set. Therefore the columns of Z are unit vectors and the reduced Hessian $Z^T GZ$ is obtained by selecting appropriate rows and columns from G. This preconditioning strategy is therefore practical and efficient since the matrix Z need not be computed and the reduced Hessian $Z^T GZ$ is easy to form.

In fact, this procedure is essentially the same as that used in TRON. The gradient projection method selects a working set (a set of active bounds) by using a gradient projection search, and computes a step by solving a quadratic program of the form (2). To solve this quadratic program, the gradient projection method in TRON eliminates the constraints and applies a preconditioned CG method to the reduced problem

$$\underset{u}{\text{minimize}} \quad u^T Z^T GZu + h^T Zu.$$

The preconditioner is defined by the incomplete Cholesky factorization (5). Thus the only difference between the CG iterations in TRON and the preconditioned projected CG method based on Algorithm PCG is that the latter works in R^n while the former works in R^{n-m}. (It is easy to see that the two approaches are equivalent and that the computational costs are very similar.)

Numerical tests of KNITRO/ACTIVE using the incomplete Cholesky preconditioner just described will be reported in a forthcoming publication. In the rest of the paper, we focus on interior-point methods and report results using various preconditioning approaches.

5 Preconditioning the Interior-Point Method

The interior-point methods implemented in KNITRO solve a sequence of barrier problems of the form

$$\underset{x,s}{\text{minimize}} \quad f(x) - \mu \sum_{i \in \mathcal{I}} \log s_i \tag{7a}$$

$$\text{subject to} \quad c_E(x) = 0 \tag{7b}$$

$$c_I(x) - s = 0, \tag{7c}$$

where s is a vector of slack variables, $\mu > 0$ is the barrier parameter, and $c_E(x), c_I(x)$ denote the equality and inequality constraints, respectively. KNITRO/CG finds an approximate solution of (7) using a form of sequential quadratic programming. This leads to an equality constrained subproblem of the form (2), in which the Hessian and Jacobian matrices are given by

$$G = \begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 \\ 0 & \Sigma \end{bmatrix}, \qquad A = \begin{bmatrix} A_E & 0 \\ A_I & -I \end{bmatrix}, \tag{8}$$

where $\mathcal{L}(x, \lambda)$ is the Lagrangian of the nonlinear program, Σ is a diagonal matrix and A_E and A_I denote the Jacobian matrices corresponding to the equality and inequality constraints, respectively. (In the bound constrained case, A_E does not exist and A_I is a simple sparse matrix whose rows are unit vectors.) The matrix Σ is defined as $\Sigma = S^{-1}\Lambda_I$, where

$$S = \text{diag}\,\{s_i\}\,, \quad \Lambda_I = \text{diag}\,\{\lambda_i\}\,, \quad i \in \mathcal{I},$$

and where the s_i are slack variables and λ_i , $i \in \mathcal{I}$ are Lagrange multipliers corresponding to the inequality constraints. Hence there are two separate sources of ill-conditioning in G; one caused by the Hessian $\nabla^2_{xx}\mathcal{L}$ and the other by the barrier effects reflected in Σ. Any ill-conditioning due to A is removed by the projected CG approach.

Given the block structure (8), the preconditioning operation (3) takes the form

$$\begin{bmatrix} D_x & 0 & {A_E}^T & {A_I}^T \\ 0 & D_s & 0 & -I \\ A_E & 0 & 0 & 0 \\ A_I & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} z_x \\ z_s \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ 0 \\ 0 \end{bmatrix}. \tag{9}$$

The matrix D_s will always be chosen as a diagonal matrix, given that Σ is diagonal. In the experiments reported in Table 1, KNITRO/CG was implemented with $D_x = I$ and $D_s = S^{-2}$. This means that the algorithm does not include preconditioning for the Hessian $\nabla^2_{xx}\mathcal{L}$, and applies a form of preconditioning for the barrier term Σ (as we discuss below). The high computing times of KNITRO/CG in Table 1 indicate that this preconditioning strategy is not effective for many problems, and therefore we discuss how to precondition each of the two terms in G.

5.1 Hessian Preconditioning

Possible preconditioners for the Hessian $\nabla^2_{xx}\mathcal{L}$ include diagonal preconditioning and incomplete Cholesky. Diagonal preconditioners are simple to

implement; we report results for them in the next section. To design an incomplete Cholesky preconditioner, we exploit the special structure of (9).

Performing block elimination on (9) yields the condensed system

$$\begin{bmatrix} D_x + {A_I}^T D_s A_I & {A_E}^T \\ A_E & 0 \end{bmatrix} \begin{bmatrix} z_x \\ w_1 \end{bmatrix} = \begin{bmatrix} r_1 + {A_I}^T r_2 \\ 0 \end{bmatrix}; \tag{10}$$

the eliminated variables z_s, w_2 are recovered from the relation

$$z_s = A_I z_x, \quad w_2 = D_s z_s - r_2.$$

If we define $D_x = LL^T$, where L is the incomplete Cholesky factor of $\nabla^2\mathcal{L}$, we still have to face the problem of how to factor (10) efficiently.

However, for problems without equality constraints, such as bound constrained problems, (10) reduces to

$$(D_x + {A_I}^T D_s A_I) z_x = r_1 + {A_I}^T r_2. \tag{11}$$

Let us assume that the diagonal preconditioning matrix D_s is given. For bound constrained problems, ${A_I}^T D_s A_I$ can be expressed as the sum of two diagonal matrices. Hence, the coefficient matrix in (11) is easy to form. Setting $D_x = \nabla^2_{xx}\mathcal{L}$, we compute the (possibly modified) incomplete Cholesky factorization

$$LL^T \approx \nabla^2_{xx}\mathcal{L} + {A_I}^T D_s A_I. \tag{12}$$

The preconditioning step is then obtained by solving

$$LL^T z_x = r_1 + {A_I}^T r_2 \tag{13}$$

and by defining $z_s = A_I z_x$.

One advantage of this approach is apparent from the structure of the matrix in the right hand side of (12). Since we are adding a positive diagonal matrix to $\nabla^2_{xx}\mathcal{L}$, it is less likely that a modification of the form δI must be introduced in the course of the incomplete Cholesky factorization. Minimizing the use of the modification δI is desirable because it can introduce undesirable distortions in the Hessian information. We note that the incomplete factorization (12) is also practical for problems that contain general inequality constraints, provided the term ${A_I}^T D_s A_I$ is not costly to form and does not lead to severe fill-in.

5.2 Barrier Preconditioning

It is well known that the matrix $\Sigma = S^{-1}\Lambda_I$ becomes increasingly ill-conditioned as the iterates of the optimization algorithm approach the solution. Some diagonal elements of Σ diverge while others converge to zero. Since Σ is a diagonal matrix, it can always be preconditioned adequately using a diagonal matrix. We consider two preconditioners:

$$D_s = \Sigma \qquad \text{and} \qquad D_s = \mu S^{-2}.$$

The first is the natural choice corresponding to the perfect preconditioner for the barrier term, while the second choice is justified because near the central path, $\Lambda_I \approx \mu S^{-1}$, so $\Sigma = S^{-1}\Lambda_I \approx S^{-1}(\mu S^{-1}) = \mu S^{-2}$.

5.3 Numerical Results

We test the preconditioners discussed above using a MATLAB implementation of the algorithm in KNITRO/CG. Our MATLAB program does not contain all the features of KNITRO/CG, but is sufficiently robust and efficient to study the effectiveness of various preconditioners.

The results are given by Table 2, which reports the preconditioning option (`option`), the final objective function value, the number of iterations of the interior-point algorithm, the total number of CG iterations, the average number of CG iterations per interior-point iteration, and the CPU time. The preconditioning options are labeled as:

$$\text{option} = (a, b)$$

where a denotes the Hessian preconditioner and b the barrier preconditioner. The options are:

$a = 0$:	No Hessian preconditioning (current default in KNITRO)
$a = 1$:	Diagonal Hessian preconditioning
$a = 2$:	Incomplete Cholesky preconditioning
$b = 0$:	$D_s = S^{-2}$ (current default in KNITRO)
$b = 1$:	$D_s = \mu S^{-2}$
$b = 2$:	$D_s = \Sigma$.

Since our MATLAB code is not optimized for speed, we have chosen test problems with a relatively small number of variables.

problem	option	final objective	#iteration	#total CG	#average CG	time
biggsb1	(0,0)	$+1.5015971301e-02$	31	3962	$1.278e+02$	$3.226e+01$
($n = 100$)	(0,1)	$+1.5015971301e-02$	29	2324	$8.014e+01$	$1.967e+01$
	(0,2)	$+1.5015971301e-02$	28	2232	$7.971e+01$	$1.880e+01$
	(1,0)	$+1.5015971301e-02$	30	3694	$1.231e+02$	$3.086e+01$
	(1,1)	$+1.5015971301e-02$	30	2313	$7.710e+01$	$2.010e+01$
	(1,2)	$+1.5015971301e-02$	30	2241	$7.470e+01$	$2.200e+01$
	(2,0)	$+1.5015971301e-02$	31	44	$1.419e+00$	$1.950e+00$
	(2,1)	$+1.5015971301e-02$	29	42	$1.448e+00$	$1.870e+00$
	(2,2)	$+1.5015971301e-02$	28	41	$1.464e+00$	$1.810e+00$
cvxbqp1	(0,0)	$+9.0450040000e+02$	11	91	$8.273e+00$	$4.420e+00$
($n = 200$)	(0,1)	$+9.0453998374e+02$	8	112	$1.400e+01$	$4.220e+00$
	(0,2)	$+9.0450040000e+02$	53	54	$1.019e+00$	$1.144e+01$
	(1,0)	$+9.0454000245e+02$	30	52	$1.733e+00$	$9.290e+00$
	(1,1)	$+9.0450040000e+02$	30	50	$1.667e+00$	$9.550e+00$
	(1,2)	$+9.0454001402e+02$	47	48	$1.021e+00$	$1.527e+01$
	(2,0)	$+9.0450040000e+02$	11	18	$1.636e+00$	$2.510e+00$
	(2,1)	$+9.0454000696e+02$	8	15	$1.875e+00$	$1.940e+00$
	(2,2)	$+9.0450040000e+02$	53	53	$1.000e+00$	$1.070e+01$
jnlbrng1	(0,0)	$-1.7984674056e-01$	29	5239	$1.807e+02$	$8.671e+01$
($n = 324$)	(0,1)	$-1.7984674056e-01$	27	885	$3.278e+01$	$1.990e+01$
	(0,2)	$-1.7984674056e-01$	29	908	$3.131e+01$	$2.064e+01$
	(1,0)	$-1.7984674056e-01$	29	5082	$1.752e+02$	$9.763e+01$
	(1,1)	$-1.7984674056e-01$	27	753	$2.789e+01$	$3.387e+01$
	(1,2)	$-1.7988019171e-01$	26	677	$2.604e+01$	$2.917e+01$
	(2,0)	$-1.7984674056e-01$	30	71	$2.367e+00$	$6.930e+00$
	(2,1)	$-1.7984674056e-01$	27	59	$2.185e+00$	$6.390e+00$
	(2,2)	$-1.7984674056e-01$	29	66	$2.276e+00$	$6.880e+00$
obstclbm	(0,0)	$+5.9472925926e+00$	28	7900	$2.821e+02$	$1.919e+02$
($n = 225$)	(0,1)	$+5.9473012340e+00$	18	289	$1.606e+01$	$1.268e+01$
	(0,2)	$+5.9472925926e+00$	31	335	$1.081e+01$	$1.618e+01$
	(1,0)	$+5.9472925926e+00$	27	6477	$2.399e+02$	$1.620e+02$
	(1,1)	$+5.9472925926e+00$	29	380	$1.310e+01$	$2.246e+01$
	(1,2)	$+5.9473012340e+00$	18	197	$1.094e+01$	$1.192e+01$
	(2,0)	$+5.9472925926e+00$	27	49	$1.815e+00$	$7.180e+00$
	(2,1)	$+5.9473012340e+00$	17	32	$1.882e+00$	$4.820e+00$
	(2,2)	$+5.9472925926e+00$	25	49	$1.960e+00$	$6.650e+00$
pentdi	(0,0)	$-7.4969998494e-01$	27	260	$9.630e+00$	$6.490e+00$
($n = 250$)	(0,1)	$-7.4969998502e-01$	25	200	$8.000e+00$	$5.920e+00$
	(0,2)	$-7.4969998500e-01$	28	205	$7.321e+00$	$5.960e+00$
	(1,0)	$-7.4969998494e-01$	28	256	$9.143e+00$	$1.111e+01$
	(1,1)	$-7.4992499804e-01$	23	153	$6.652e+00$	$9.640e+00$
	(1,2)	$-7.4969998502e-01$	26	132	$5.077e+00$	$9.370e+00$
	(2,0)	$-7.4969998494e-01$	27	41	$1.519e+00$	$3.620e+00$
	(2,1)	$-7.4969998502e-01$	25	39	$1.560e+00$	$3.350e+00$
	(2,2)	$-7.4969998500e-01$	28	42	$1.500e+00$	$3.640e+00$
torsion1	(0,0)	$-4.8254023392e-01$	26	993	$3.819e+01$	$9.520e+00$
($n = 100$)	(0,1)	$-4.8254023392e-01$	25	298	$1.192e+01$	$4.130e+00$
	(0,2)	$-4.8254023392e-01$	24	274	$1.142e+01$	$3.820e+00$
	(1,0)	$-4.8254023392e-01$	26	989	$3.804e+01$	$9.760e+00$
	(1,1)	$-4.8254023392e-01$	25	274	$1.096e+01$	$4.520e+00$
	(1,2)	$-4.8254023392e-01$	25	250	$1.000e+01$	$3.910e+00$
	(2,0)	$-4.8254023392e-01$	25	52	$2.080e+00$	$1.760e+00$
	(2,1)	$-4.8254023392e-01$	25	53	$2.120e+00$	$1.800e+00$
	(2,2)	$-4.8254023392e-01$	24	51	$2.125e+00$	$1.660e+00$
torsionb	(0,0)	$-4.0993481087e-01$	25	1158	$4.632e+01$	$1.079e+01$
($n = 100$)	(0,1)	$-4.0993481087e-01$	25	303	$1.212e+01$	$4.160e+00$
	(0,2)	$-4.0993481087e-01$	23	282	$1.226e+01$	$3.930e+00$
	(1,0)	$-4.0993481087e-01$	25	1143	$4.572e+01$	$1.089e+01$
	(1,1)	$-4.0993481087e-01$	24	274	$1.142e+01$	$4.450e+00$
	(1,2)	$-4.0993481087e-01$	23	246	$1.070e+01$	$3.700e+00$
	(2,0)	$-4.0993481087e-01$	24	49	$2.042e+00$	$1.720e+00$
	(2,1)	$-4.0993481087e-01$	24	49	$2.042e+00$	$1.700e+00$
	(2,2)	$-4.0993481087e-01$	23	48	$2.087e+00$	$1.630e+00$

Table 2. Results of various preconditioning options

Note that for all the test problems, except `cvxbqp1`, the number of interior-point iterations is not greatly affected by the choice of preconditioner. Therefore, we can use Table 2 to measure the efficiency of the preconditioners, but we must exercise caution when interpreting the results for problem `cvxbqp1`.

Let us consider first the case when only barrier preconditioning is used, i.e., where `option` has the form $(0, *)$. As expected, the options $(0, 1)$ and $(0, 2)$ generally decrease the number of CG iterations and computing time with respect to the standard option $(0, 0)$, and can therefore be considered successful in this context. From these experiments it is not clear whether option $(0, 1)$ is to be preferred over option $(0, 2)$.

Incomplete Cholesky preconditioning is very successful. If we compare the results for options (0,0) and (2,0), we see substantial reductions in the number of CG iterations and computing time for the latter option. When we add barrier preconditioning to incomplete Cholesky preconditioning (options $(2, 1)$ and $(2, 2)$) we do not see further gains. Therefore, we speculate that the standard barrier preconditioner $D_s = S^{-2}$ may be adequate, provided the Hessian preconditioner is effective.

Diagonal Hessian preconditioning, i.e, options of the form $(1, *)$, rarely provides much benefit. Clearly this preconditioner is of limited use.

One might expect that preconditioning would not affect much the number of iterations of the interior-point method because it is simply a mechanism for accelerating the step computation procedure. The results for problem `cvxbqp1` suggest that this is not the case (we have seen a similar behavior on other problems). In fact, preconditioning changes the form of the algorithm in two ways: it changes the shape of the trust region and it affects the barrier stop test.

We introduce preconditioning in KNITRO/CG by defining the trust region as

$$\left\| \begin{bmatrix} D_x^{1/2} d_x \\ D_s^{1/2} d_s \end{bmatrix} \right\|_2 \leq \Delta.$$

The standard barrier preconditioner $D_s = S^{-2}$ gives rise to the trust-region

$$\left\| \begin{bmatrix} D_x^{1/2} d_x \\ S^{-1} d_s \end{bmatrix}_2 \right\| \leq \Delta, \tag{14}$$

which has proved to control well the rate at which the slacks approach zero. (This is the standard affine scaling strategy used in many optimization methods.) On the other hand, the barrier preconditioner $D_s = \mu S^{-2}$ results in the trust region

$$\left\| \begin{bmatrix} D_x^{1/2} d_x \\ \sqrt{\mu} S^{-1} d_s \end{bmatrix} \right\|_2 \leq \Delta. \tag{15}$$

When μ is small, (15) does not penalize a step approaching the bounds $s \geq 0$ as severely as (14). This allows the interior-point method to approach the

boundary of the feasible region prematurely and can lead to very small steps. An examination of the results for problem `cvxbqp1` shows that this is indeed the case. The preconditioner $D_s = \Sigma = S^{-1}\Lambda_I$ can be ineffective for a different reason. When the multiplier estimates λ_i are inaccurate (too large or too small) the trust region will not properly control the step d_s.

These remarks reinforce our view that the standard barrier preconditioner $D_s = S^{-2}$ may be the best choice and that our effort should focus on Hessian preconditioning.

Let us consider the second way in which preconditioning changes the interior-point algorithm. Preconditioning amounts to a scaling of the variables of the problem; this scaling alters the form of the KKT optimality conditions. KNITRO/CG uses a barrier stop test that determines when the barrier problem has been solved to sufficient accuracy. This strategy forces the iterates to remain in a (broad) neighborhood of the central path. Each barrier problem is terminated when the norm of the scaled KKT conditions is small enough, where the scaling factors are affected by the choice of D_x and D_s. A poor choice of preconditioner, including diagonal Hessian preconditioning, introduces an unwanted distortion in the barrier stop test, and this can result in a deterioration of the interior-point iteration. Note in contrast that the incomplete Cholesky preconditioner (option $(2, *)$) does not adversely affect the overall behavior of the interior-point iteration in problem `cvxbqp1`.

6 Quasi-Newton Methods

We now consider algorithms that use quasi-Newton approximations. In recent years, most of the numerical studies of interior-point methods have focused on the use of exact Hessian information. It is well known, however, that in many practical applications, second derivatives are not available, and it is therefore of interest to compare the performance of active-set and interior-point methods in this context. We report results with 5 solvers: SNOPT version 7.2-1 [10], L-BFGS-B [4, 19], KNITRO/DIRECT, KNITRO/CG and KNITRO/ACTIVE version 5.0. Since all the problems in our test set have more than 1000 variables, we employ the limited memory BFGS quasi-Newton options in all codes, saving $m = 20$ correction pairs. All other options in the codes were set to their defaults.

SNOPT is an active-set SQP method that computes steps by solving an inequality constrained quadratic program. L-BFGS-B implements a gradient projection method. Unlike TRON, which is a trust region method, L-BFGS-B is a line search algorithm that exploits the simple structure of limited memory quasi-Newton matrices to compute the step at small cost. Table 3 reports the results on the same set of problems as in Table 1. Performance profiles are provided in Figures 3 and 4.

		SNOPT ($m = 20$)			L-BFGS-B ($m = 20$)			KNITRO/DIRECT ($m = 20$)			KNITRO/CG ($m = 20$)			KNITRO/ACTIVE ($m = 20$)		
problem	n	iter	feval	CPU	iter	feval	CPU	iter	feval	CPU	iter	feval	CPU	iter	feval	CPU
biggsb1	20000	X_1	X_1	X_1	X_1	X_1	X_1	6812	6950	1244.05	3349	3443	1192.32	X_1	X_1	X_1
bqpgauss	2003	5480	6138	482.87	9686	10253	96.18	X_1	X_1	X_1	X_1	X_1	X_1	X_4	X_4	X_4
chenhark	20000	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1
clnlbeam	20000	41	43	45.18	22	28	0.47	19	20	31.42	14	15	11.94	16	17	6.42
cvxbqp1	20000	60	65	139.31	1	2	0.04	29	30	89.03	25	26	71.44	2	3	0.59
explin	24000	72	100	28.08	29	36	0.52	50	51	239.29	47	48	76.84	32	34	35.84
explin2	24000	63	72	25.62	20	24	0.30	33	34	133.65	40	41	69.74	23	28	17.32
expquad	24000	X_4	X_4	X_4	X_2	X_2	X_2	X_4	X_4	X_4	X_5	X_5	X_5	206	645	513.75
gridgena	26312	X_6	X_6	X_6	X_7	X_7	X_7	X_5	X_5	X_5	X_5	X_5	X_5	20	97	120.59
harkerp2	2000	50	57	7.05	86	102	4.61	183	191	76.26	164	168	58.82	10	11	1.48
jnlbrng1	21904	1223	1337	8494.55	1978	1992	205.02	1873	1913	992.23	1266	1309	1968.66	505	515	1409.80
jnlbrnga	21904	1179	1346	1722.60	619	640	59.24	2134	2191	10929.97	1390	1427	221.73	395	417	1236.32
mccormck	100000	1019	1021	10820.22	X_8	X_8	X_8	53	166	1222.38	X_4	X_4	X_4	X_5	X_5	X_5
minsurfo	10000	904	1010	8712.90	1601	1648	97.66	3953	3980	801.87	1633	1665	16136.98	497	498	743.37
ncvxbqp1	20000	41	43	60.54	1	2	0.04	85	86	382.62	X_1	X_1	X_1	9	10	2.03
ncvxbqp2	20000	X_6	X_6	X_6	151	191	4.76	3831	3835	20043.27	8118	8119	993.03	124	125	178.97
nobndtor	32400	1443	1595	12429	1955	1966	314.42	1100	1129	8306.03	1049	1069	27844.21	873	886	3155.06
nonscomp	20000	233	237	1027.41	X_8	X_8	X_8	31	34	99.36	1098	1235	2812.25	87	92	123.82
obstclae	21904	547	597	4344.33	1110	1114	109.11	982	1009	1322.69	618	639	11489.74	1253	1258	2217.58
obstclbm	21904	342	376	1332.14	359	368	35.94	383	391	2139.91	282	286	1222.99	276	279	641.07
pentdi	20000	2	6	0.57	1	3	0.05	59	61	221.98	60	62	67.39	3	7	0.72
probpenl	5000	3	5	8.86	2	4	0.03	4	8	0.53	4	5	0.30	2	4	0.10
qrtquad	5000	X_6	X_6	X_6	241	308	4.85	X_4	X_4	X_4	X_5	X_5	X_5	X_1	X_1	X_1
qudlin	20000	41	43	19.80	1	2	0.02	17	18	27.78	24	25	34.81	4	5	0.43
reading1	20001	81	83	114.18	7593	15354	234.93	359	625	1891.24	66	69	150.48	15	16	5.17
scond1ls	2000	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1	X_1
sineali	20000	466	553	918.33	14	19	0.63	X_5	X_5	X_5	X_4	X_4	X_4	X_1	X_1	X_1
torsion1	32400	662	733	4940.83	565	579	86.39	696	716	1564.78	336	362	15661.85	300	303	1251.40
torsiona	32400	685	768	5634.62	490	496	77.42	625	643	950.16	349	370	15309.47	296	306	1272.50

X_1: iteration limit reached
X_2: numerical result out of range
X_4: current solution estimate cannot be improved
X_5: relative change in solution estimate $< 10^{-15}$
X_6: dual feasibility cannot be satisfied
X_7: rounding error
X_8: line search error

Table 3. Comparative results for five methods that approximate second derivative information by limited memory quasi-Newton updates

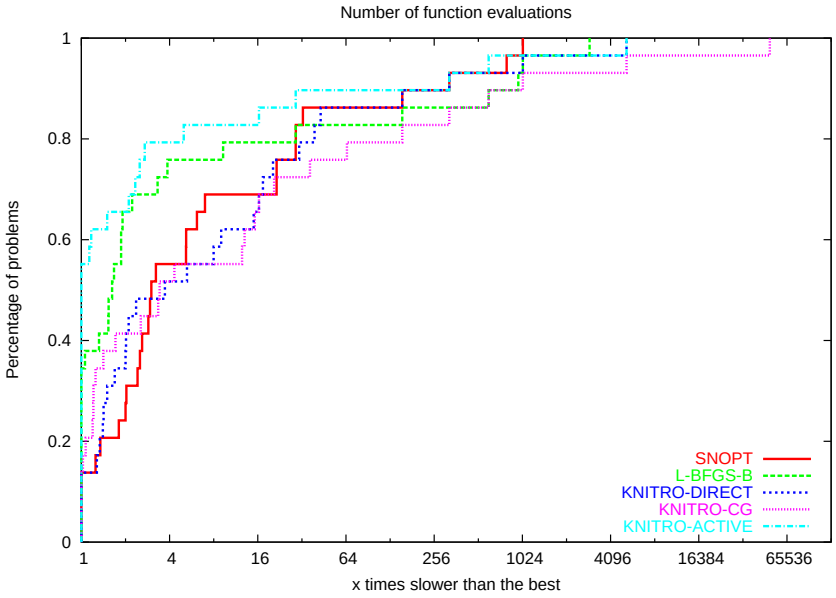

Fig. 3. Number of Function Evaluations

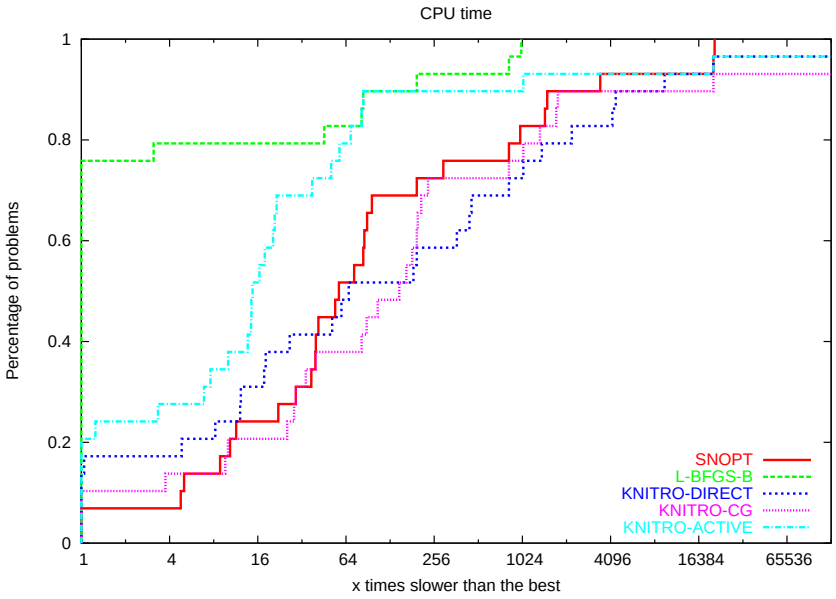

Fig. 4. CPU Time

A sharp drop in robustness and speed is noticeable for the three KNITRO algorithms; compare with Table 1. In terms of function evaluations, L-BFGS-B and KNITRO/ACTIVE perform the best. SNOPT and the two interior-point methods require roughly the same number of function evaluations, and this number is often dramatically larger than that obtained by the interior-point solvers using exact Hessian information.

In terms of CPU time, L-BFGS-B is by far the best solver and KNITRO/ACTIVE comes in second. Again, SNOPT and the two interior-point methods require a comparable amount of CPU, and for some of these problems the times are unacceptably high.

In summation, as was the case with TRON when exact Hessian information was available, the specialized quasi-Newton method for bound constrained problems L-BFGS-B has an edge over the general purpose solvers. The use of preconditioning has helped bridge the gap in the exact Hessian case, but in the quasi-Newton case, improved updating procedures are clearly needed for general purpose methods.

References

1. L. Bergamaschi, J. Gondzio, and G. Zilli, *Preconditioning indefinite systems in interior point methods for optimization*, Tech. Rep. MS-02-002, Department of Mathematics and Statistics, University of Edinburgh, Scotland, 2002.
2. R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz, *An algorithm for nonlinear optimization using linear programming and equality constrained subproblems*, Mathematical Programming, Series B, 100 (2004), pp. 27–48.
3. R. H. Byrd, M. E. Hribar, and J. Nocedal, *An interior point algorithm for large scale nonlinear programming*, SIAM Journal on Optimization, 9 (1999), pp. 877–900.
4. R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM Journal on Scientific Computing, 16 (1995), pp. 1190–1208.
5. R. H. Byrd, J. Nocedal, and R. Waltz, *KNITRO: An integrated package for nonlinear optimization*, in Large-Scale Nonlinear Optimization, G. di Pillo and M. Roma, eds., Springer, 2006, pp. 35–59.
6. R. H. Byrd and R. A. Waltz, *Improving SLQP methods using parametric linear programs*, tech. rep., OTC, 2006. To appear.
7. E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming, Series A, 91 (2002), pp. 201–213.
8. A. Forsgren, P. E. Gill, and J. D. Griffin, *Iterative solution of augmented systems arising in interior methods*, Tech. Rep. NA 05-3, Department of Mathematics, University of California, San Diego, 2005.
9. R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Scientific Press, 1993. `www.ampl.com`.
10. P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, SIAM Journal on Optimization, 12 (2002), pp. 979–1006.

11. N. I. M. Gould, D. Orban, and P. L. Toint, CUTEr *and* sifdec*: A Constrained and Unconstrained Testing Environment, revisited*, ACM Trans. Math. Softw., 29 (2003), pp. 373–394.
12. N. I. M. Gould, D. Orban, and P. L. Toint, *Numerical methods for large-scale nonlinear optimization*, Technical Report RAL-TR-2004-032, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2004.
13. C. Keller, N. I. M. Gould, and A. J. Wathen, *Constraint preconditioning for indefinite linear systems*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1300–1317.
14. C. J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM Journal on Scientific Computing, 21 (1999), pp. 24–45.
15. ———, *Newton's method for large bound-constrained optimization problems*, SIAM Journal on Optimization, 9 (1999), pp. 1100–1127.
16. M. Roma, *Dynamic scaling based preconditioning for truncated Newton methods in large scale unconstrained optimization: The complete results*, Technical Report R. 579, Istituto di Analisi dei Sistemi ed Informatica, 2003.
17. T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM Journal on Numerical Analysis, 20 (1983), pp. 626–637.
18. R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban, *An interior algorithm for nonlinear optimization that combines line search and trust region steps*, Mathematical Programming, Series A, 107 (2006), pp. 391–408.
19. 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 (1997), pp. 550–560.