An Interior Algorithm for Nonlinear Optimization That Combines Line Search and Trust Region Steps

R.A. Waltz* J.L. Morales† J. Nocedal* D. Orban*

September 8, 2004

Abstract

An interior-point method for nonlinear programming is presented. It enjoys the flexibility of switching between a line search method that computes steps by factoring the primal-dual equations and a trust region method that uses a conjugate gradient iteration. Steps computed by direct factorization are always tried first, but if they are deemed ineffective, a trust region iteration that guarantees progress toward stationarity is invoked. To demonstrate its effectiveness, the algorithm is implemented in the KNITRO [6, 28] software package and is extensively tested on a wide selection of test problems.

1 Introduction

In this paper we describe an interior method for nonlinear programming and discuss its software implementation and numerical performance. A typical iteration computes a primary step by solving the primal-dual equations (using direct linear algebra) and performs a line search to ensure decrease in a merit function. In order to obtain global convergence in the presence of nonconvexity and Hessian or Jacobian rank deficiencies, the primary step is replaced, under certain circumstances, by a safeguarding trust region step. The algorithm can use exact second derivatives of the objective function and constraints or quasi-Newton approximations.

The motivation for this paper is to develop a new interior point algorithm, implemented in the KNITRO software package [28], which is more robust and efficient than either a pure trust region or a pure line search interior approach. The algorithm implemented in the first release of KNITRO [6] is a trust region method that uses a null-space decomposition and a projected conjugate gradient iteration to compute the step. This iterative approach has the advantage that the Hessian of the Lagrangian need not be formed or factored, which is

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

†Departamento de Matemáticas, ITAM, México. This author was supported by Asociación Mexicana de Cultura, A.C. and CONACyT grant 39372-A.

effective for many large problems. The disadvantage is that the projected conjugate gradient iteration can be expensive when the Hessian of the Lagrangian of the nonlinear program is ill-conditioned, in which case it is preferable to use direct linear algebra techniques to compute the step.

By designing the algorithm so that it computes steps using direct linear algebra whenever the quality of these steps can be guaranteed, and falling back on a trust region step otherwise, we can easily implement the new method within the existing KNITRO package. Moreover, since the algorithm can reduce to a pure line search or a pure trust region approach, we are able to implement two different interior methods in a unified algorithmic and software framework.

The problem under consideration has the form

$$\min_x \quad f(x) \tag{1.1a}$$

$$\text{s.t.} \quad h(x) = 0 \tag{1.1b}$$

$$g(x) \leq 0, \tag{1.1c}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^l$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable functions. A variety of line search interior algorithms have been proposed [12, 16, 25, 27, 30], several of which have been implemented in high-quality software packages [2, 25, 27]. The search direction is computed in these algorithms by factoring the primal-dual system. In order to achieve robustness, these line search approaches must successfully address two issues:

- How to define the search direction when the quadratic model used by the algorithm is not convex;
- How to handle rank deficiency (and near deficiency) of the Hessian of the Lagrangian and constraint Jacobian.

The first issue is often addressed by adding a multiple of the identity matrix to the Hessian of the Lagrangian in order to convexify the model. This strategy was first shown to be effective in the context of nonlinear interior methods by the LOQO software package [25]. The second issue is handled differently in each software implementation. The difficulties caused by rank deficient constraint Jacobians are sometimes addressed at the linear algebra level by introducing perturbations during the factorization of the KKT matrix [1]. Other approaches include the use of ℓ_1 or ℓ_2 penalizations of the constraints, which provide regularization [16, 24], and the use of a feasibility restoration phase [15, 27].

In this paper we describe a mechanism for stabilizing the line search iteration that is different from those proposed in the literature. It consists of falling back, under certain conditions, on a trust region step that is guaranteed to make progress toward feasibility and optimality. A challenge is to design the algorithm so that there is a smooth transition between line search and trust region steps. We will argue that the algorithm presented in this paper is not more expensive than other approaches, has favorable convergence properties, and performs well on standard test problems.

Notation. Throughout the paper $\|\cdot\|$ denotes the Euclidean norm.

2 Outline of the Algorithm

We consider an interior method that replaces the nonlinear program (1.1) by a sequence of barrier subproblems of the form

$$\min_z \quad \varphi_\mu(z) \equiv f(x) - \mu \sum_{i=1}^m \ln s_i \tag{2.1a}$$

$$\text{s.t.} \quad h(x) = 0 \tag{2.1b}$$

$$g(x) + s = 0. \tag{2.1c}$$

Here $s > 0$ is a vector of slack variables, $z = (x, s)$, and $\mu > 0$ is the barrier parameter. The Lagrangian function associated with (2.1) is defined by

$$\mathcal{L}(z, \lambda; \mu) = \varphi_\mu(z) + \lambda_h^T h(x) + \lambda_g^T (g(x) + s), \tag{2.2}$$

where $\lambda_h \in \mathbb{R}^l$ and $\lambda_g \in \mathbb{R}^m$ are Lagrange multipliers and $\lambda = (\lambda_h, \lambda_g)$. The first-order optimality conditions for the barrier problem (2.1) can be written as

$$\begin{bmatrix} \nabla f(x) + A_h(x)^T \lambda_h + A_g(x)^T \lambda_g \\ S\Lambda_g e - \mu e \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{2.3}$$

together with (2.1b) and (2.1c) and the restriction that s and λ_g be nonnegative. Here S and Λ_g denote diagonal matrices whose diagonal entries are given by the vectors s and λ_g, respectively, and A_h and A_g are the Jacobian matrices of h and g.

Applying Newton's method to the system (2.3), (2.1b), (2.1c), from the current iterate (z, λ) results in the primal-dual system

$$\begin{bmatrix} W(z, \lambda; \mu) & A(x)^T \\ A(x) & 0 \end{bmatrix} \begin{bmatrix} d_z \\ d_\lambda \end{bmatrix} = - \begin{bmatrix} \nabla_z \mathcal{L}(z, \lambda; \mu) \\ c(z) \end{bmatrix}, \tag{2.4}$$

where we have defined

$$d_z = \begin{bmatrix} d_x \\ d_s \end{bmatrix}, \quad d_\lambda = \begin{bmatrix} d_h \\ d_g \end{bmatrix}, \quad c(z) = \begin{bmatrix} h(x) \\ g(x) + s \end{bmatrix} \tag{2.5}$$

and

$$A(x) = \begin{bmatrix} A_h(x) & 0 \\ A_g(x) & I \end{bmatrix}, \quad W(z, \lambda; \mu) = \begin{bmatrix} \nabla^2_{xx} \mathcal{L}(z, \lambda; \mu) & 0 \\ 0 & S^{-1}\Lambda_g \end{bmatrix}. \tag{2.6}$$

The new iterate is given by

$$z^+ = z + \alpha_z d_z, \quad \lambda^+ = \lambda + \alpha_\lambda d_\lambda. \tag{2.7}$$

The steplengths α_z and α_λ are computed in two stages. First we compute

$$\alpha_z^{\max} = \max\{\alpha \in (0, 1] : s + \alpha d_s \geq (1 - \tau)s\} \tag{2.8a}$$

$$\alpha_\lambda^{\max} = \max\{\alpha \in (0, 1] : \lambda_g + \alpha d_g \geq (1 - \tau)\lambda_g\}, \tag{2.8b}$$

with $0 < \tau < 1$ and typically close to one. (In our tests we use $\tau = .995$.) The algorithm then performs a backtracking line search that computes steplengths

$$\alpha_z \in (0, \alpha_z^{\max}], \qquad \alpha_\lambda \in (0, \alpha_\lambda^{\max}], \tag{2.9}$$

providing sufficient decrease in a merit function (to be defined below). The procedure for updating the barrier parameter μ is described in Section 3.5.

The iteration (2.4)-(2.7) provides the basis for most line search interior methods. This remarkably simple approach must, however, be modified to cope with non-convexity and to prevent convergence to non-stationary points [7, 26]. Instead of modifying the primal-dual matrix, as is commonly done, we use a safeguarding trust region step to stabilize the iteration, for two reasons. First, when W is not positive definite on the null-space of A (the negative curvature case), adding to W a multiple of the identity matrix may introduce undesirable distortions in the model and can also require several factorizations of the primal-dual system. In the negative curvature case, we prefer to compute a descent direction using a null space approach in which the tangential component of the step is obtained by using a projected Krylov iteration, as is done with conjugate gradients in the KNITRO package [28], or using a Lanczos method, as is done in the GALAHAD package [19].

Second, we would like to take advantage of the robustness of trust region steps in the presence of Hessian or Jacobian rank deficiencies. We have in mind trust region methods that provide Cauchy decrease for both feasibility and optimality at every iteration. Since it is known that, when line search iterations converge to non-stationary points, the steplengths α_z or α_λ in (2.7) converge to zero, we monitor these steplengths. If they become smaller than a given threshold, we discard the line search iteration (2.4)-(2.7) and replace it with a trust region step. The resulting algorithm possesses global convergence properties similar to those of the algorithms implemented in FILTERSQP [15] and KNITRO [5].

We outline the method in Algorithm 2.1. There $\phi_\nu(z)$ denotes a merit function using a penalty parameter ν, and $D\phi_\nu(z; d_z)$ denotes the directional derivative of ϕ_ν along a direction d_z. An inertia-revealing symmetric indefinite factorization [4] of the primal-dual matrix in (2.4) provides its number of negative eigenvalues. If this number exceeds $l + m$, then d_z cannot be guaranteed to be a descent direction (see, e.g., [21, Lemma 16.3]), and the primal-dual step is discarded.

Algorithm 2.1 Outline of the Interior Algorithm
Choose $z_0 = (x_0, s_0)$ *and the parameters* $0 < \eta$, $0 < \delta < 1$ *and an integer* `imax`. *Compute initial values for the multipliers* λ_0, *the trust-region radius* $\Delta_0 > 0$, *and the barrier parameter* $\mu_0 > 0$. *Set* $k = 0$.

Repeat *until a stopping test for the nonlinear program (1.1) is satisfied*
    Repeat *until a termination test for the barrier problem (2.1) is satisfied*
        Factor the primal-dual system (2.4) and record the number `neig`
        of negative eigenvalues of its coefficient matrix.
        Set `LineSearch = False`.
        If `neig` $\leq l + m$
            Solve (2.4) to obtain the search direction $d = (d_z, d_\lambda)$.
            Compute $\alpha_z^{max}, \alpha_\lambda^{max}$ *using* (2.8*a*),(2.8*b*).
            If $\min\{\alpha_z^{max}, \alpha_\lambda^{max}\} > \delta$
                Set $j = 0$, $\alpha_T = 1$.
                Repeat *while* ($j \leq$ `imax`), $(\alpha_T > \delta)$ *and* `LineSearch == False`
                    If $\phi_\nu(z_k + \alpha_T \alpha_z^{max} d_z) \leq \phi_\nu(z_k) + \eta \alpha_T \alpha_z^{max} D\phi_\nu(z_k; d_z)$
                        Set $\alpha_z = \alpha_T \alpha_z^{max}$, $\alpha_\lambda = \alpha_T \alpha_\lambda^{max}$.
                        Set z_{k+1}, λ_{k+1} *using* (2.7).
                        Compute Δ_{k+1} *and set* `LineSearch = True`.
                    Else
                        Set $j \leftarrow j + 1$.
                        Choose a smaller value of α_T.
                    Endif
                End Repeat
            Endif
        Endif
        If `LineSearch == False`
            Compute (z_{k+1}, λ_{k+1}) *using a globally convergent safeguarding*
            trust region method.
            Compute Δ_{k+1}.
        Endif
        Set $\mu_{k+1} \leftarrow \mu_k$.
        Set $k \leftarrow k + 1$.
    End Repeat
    Reset the barrier parameter μ_k *so that* $\mu_k < \mu_{k-1}$.
End Repeat

In our tests we choose $\eta = 10^{-8}$, $\delta = 10^{-5}$, and `imax`=3. The initial multipliers λ_0 are computed by least squares. When the line search step is discarded (the last If-Endif block), we compute one or more trust region steps until one of them provides sufficient reduction in the merit function. If the first steplength is not acceptable, we may, under certain circumstances, compute a second-order correction step before beginning the backtracking line search. This is described in Section 3.2.

One could consider an algorithm that, instead of switching between the two methods, would follow a dog-leg approach. Cauchy and Newton steps would be computed at every iteration, and the algorithm would move along a direction in the span of these two steps. We have not followed such an approach for two reasons. First, computing a Cauchy step that ensures progress toward feasibility and optimality requires the factorization of a system different from the primal-dual matrix [5, 30]. Hence computing Cauchy and Newton steps at every iteration is too expensive, and one should resort to the computation of Cauchy steps only when needed. Second, a dog-leg approach is not well defined in the case of negative curvature, where a Newton-CG iteration [23] is more appropriate. These observations and the fact that the first release of KNITRO implements a Newton-CG iteration motivated us to follow the approach just outlined.

Many details of Algorithm 2.1 have not been specified and will be discussed next.

3 Complete Description of the Algorithm

In this section we discuss in detail the important aspects of the algorithm excluding the safeguarding trust region steps, which are described in [6] and are computed in practice with KNITRO. Therefore, throughout this section we will focus mainly on the line search steps.

3.1 Merit Function

The merit function is defined by

$$\phi_\nu(z) = \varphi_\mu(z) + \nu\|c(z)\|, \tag{3.1}$$

where φ_μ is the barrier function defined in (2.1a), the constraints $c(z)$ are given by (2.5), and $\nu > 0$ is the penalty parameter, which is updated at each iteration so that the search direction d_z given by (2.4) is a descent direction for ϕ_ν.

Our update rule for ν, proposed in [13], is inspired by trust region methods. Instead of requiring only that the directional derivative of ϕ_ν be negative, as is commonly done, we choose ν based on the decrease in a quadratic/linear model of the merit function achieved by the step d.

In trust region methods, a step d is acceptable if the ratio of actual to predicted reduction of the merit function is greater than a given constant $\eta > 0$, that is,

$$\frac{ared(d)}{pred(d)} > \eta.$$

In this paper we define

$$ared(d_z) = \phi_\nu(z) - \phi_\nu(z + d_z)$$

and

$$pred(d_z) = -\nabla\varphi_\mu(z)^T d_z - \frac{\sigma}{2} d_z^T W d_z + \nu \left(\|c(z)\| - \|c(z) + A(x)d_z\|\right), \tag{3.2}$$

with

$$\sigma = \begin{cases} 1 & \text{if } d_z^T W d_z > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Here W stands for $W(z, \lambda; \mu)$ and is defined in (2.6). If $\sigma = 1$, *pred* is the standard quadratic/linear model of the merit function used in a variety of trust region methods; see [9]. We allow σ to have the value zero because, as we argue below, including the term $d_z^T W d_z$ in (3.2) when it is negative could cause the algorithm to fail.

Following [6, 13], we choose the penalty parameter ν so that

$$pred(d_z) \geq \rho\nu\|c(z)\|, \tag{3.4}$$

for some parameter $\rho \in (0, 1)$. (In our tests we use the value $\rho = 0.1$.) From (3.2), (3.4), and the fact that the step d_z computed from (2.4) satisfies $c(z) + A(x)d_z = 0$, we get that

$$\nu \geq \frac{\nabla\varphi_\mu(z)^T d_z + \frac{\sigma}{2} d_z^T W d_z}{(1-\rho)\|c(z)\|} \equiv \nu_{\text{TRIAL}}. \tag{3.5}$$

The update rule for the penalty parameter ν is

$$\nu^+ = \begin{cases} \nu & \text{if } \nu \geq \nu_{\text{TRIAL}} \\ \nu_{\text{TRIAL}} + 1 & \text{otherwise.} \end{cases} \tag{3.6}$$

To show that this choice of ν guarantees that d_z is a descent direction for ϕ_ν, we note that (3.5) and the definition of σ imply that

$$\nabla\varphi_\mu^T d_z - \nu\|c(z)\| \leq -\rho\nu\|c(z)\|.$$

Since the directional derivative of ϕ_ν along d_z is given by

$$D\phi_\nu(z; d_z) = \nabla\varphi_\mu^T d_z - \nu\|c(z)\| \tag{3.7}$$

(see, for example, [21, p. 545]) we have that

$$D\phi_\nu(z; d_z) \leq -\rho\nu\|c(z)\|. \tag{3.8}$$

Note, however, that this argument would not hold if σ were defined as 1 when $d_z^T W d_z < 0$, for then d_z might not be a descent direction for the merit function. (When $c(z) = 0$, we can set $\nu^+ = \nu$ because it is easy to show that in this case $D\phi_\nu(z; d) < 0$.)

We have experimented with other update procedures for ν that directly impose (3.8). One of them is given by (3.5) but with σ always equal to 0 (see, for example, [9]). We have observed that the rule described in this section, which may include curvature information about the Lagrangian, gives consistently better results. Note that including the term $d_z^T W d_z$ in (3.5) leads to larger estimates of the penalty parameter than when this term is not included.

3.2 Line Search and Second-Order Correction

After computing the primal-dual direction (d_z, d_λ) defined by (2.4), we determine the steplengths to the boundary, $\alpha_z^{\max}$ and $\alpha_\lambda^{\max}$ satisfying the inequalities (2.8*a*)-(2.8*b*). If both are greater than the threshold δ, then we perform a backtracking line search that generates a series of steplengths (stored in the parameter α_{T} in Algorithm 2.1) until one of the following two conditions is satisfied:

1. One of the steplengths $\alpha_{\mathrm{T}} < \delta$, or the maximum number `imax` of backtracks is reached. In this case, the line search is aborted, the primal-dual direction $d = (d_z, d_\lambda)$ is discarded, and the algorithm invokes the safeguarding trust region method. In our practical implementation we set `imax` $= 3$.

2. A steplength α_{T} satisfies the Armijo condition

$$\phi_\nu(z + \alpha_{\mathrm{T}}\alpha_z^{\max} d_z) \leq \phi_\nu(z) + \eta\alpha_{\mathrm{T}}\alpha_\lambda^{\max} D\phi_\nu(z; d_z). \tag{3.9}$$

   In this case, the line search terminates successfully, and we define $\alpha_z = \alpha_{\mathrm{T}}\alpha_z^{\max}$ and $\alpha_\lambda = \alpha_{\mathrm{T}}\alpha_\lambda^{\max}$.

The backtracking line search proceeds as follows. For $j = 0$ we set $\alpha_{\mathrm{T}} = 1$ as indicated in Algorithm 2.1. For $j = 1$, if the previous iteration was a line search iteration, we set $\alpha_{\mathrm{T}} = 1/2$; otherwise, if it was a trust region iteration, we set

$$\alpha_{\mathrm{T}} = \min\left(\frac{1}{2}, \frac{\Delta}{\|d_z\|}\right),$$

where Δ is the current trust region radius. For $j > 1$, the new trial steplength is updated by $\alpha_{\mathrm{T}} \leftarrow \alpha_{\mathrm{T}}/2$.

The merit function (3.1) may reject steps that make good progress toward the solution, a phenomenon known as the Maratos effect. This deficiency can be overcome by applying a second-order correction step (SOC), which is a Newton-like step that aims to improve feasibility [14].

We apply the second-order correction when the first trial steplength $\alpha_{\mathrm{T}} = 1$ is rejected and if the reason for the rejection can be attributed solely to an increase in the norm of the constraints, that is, the second term in (3.1). More specifically, if the Armijo condition (3.9) is not satisfied for $\alpha_{\mathrm{T}} = 1$, and if $\varphi_\mu(z + \alpha_z^{\max} d_z) \leq \varphi_\mu(z)$, then before computing a shorter steplength we attempt a second-order correction step $d^{\mathrm{soc}} = (d_z^{\mathrm{soc}}, d_\lambda^{\mathrm{soc}})$ by solving

$$\begin{bmatrix} W & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} d_z^{\mathrm{soc}} \\ d_\lambda^{\mathrm{soc}} \end{bmatrix} = \begin{bmatrix} -\nabla\varphi_\mu(z) - \alpha_z^{\max} W d_z - A^T[\lambda + \alpha_\lambda^{\max} d_\lambda] \\ -c(z + \alpha_z^{\max} d_z) \end{bmatrix}, \tag{3.10}$$

where $A = A(x)$ and $W = W(z, \lambda; \mu)$. Using it, we define the combined step

$$\bar{d}_z = \alpha_z^{\max} d_z + d_z^{\mathrm{soc}}, \quad \bar{d}_\lambda = \alpha_\lambda^{\max} d_\lambda + d_\lambda^{\mathrm{soc}}, \tag{3.11}$$

and let $\bar{d}_z = (\bar{d}_x, \bar{d}_s)$ and $\bar{d}_\lambda = (\bar{d}_h, \bar{d}_g)$. Next, we determine the largest positive scalars $\gamma_z^{\rm soc}, \gamma_\lambda^{\rm soc} \in (0, 1]$, such that

$$\begin{aligned} s + \gamma_z^{\rm soc} \bar{d}_s &\geq (1-\tau)s && (3.12a) \\ \lambda_g + \gamma_\lambda^{\rm soc} \bar{d}_g &\geq (1-\tau)\lambda_g, && (3.12b) \end{aligned}$$

where τ is the same constant as in (2.8), and we define

$$z^{\rm soc} = z + \gamma_z^{\rm soc} \bar{d}_z, \quad \lambda^{\rm soc} = \lambda + \gamma_\lambda^{\rm soc} \bar{d}_\lambda.$$

If $\phi_\nu(z^{\rm soc}) < \phi_\nu(z)$, then we accept the second-order correction step. Otherwise, we discard $d^{\rm soc}$ and continue the backtracking line search along the primal-dual step (d_z, d_λ) by selecting a new steplength (case $j = 1$ in Algorithm 2.1).

Note that in the left-hand side of (3.10), the matrices A and W are unchanged when compared to the system from which d itself was computed—they are not evaluated at $z + \alpha_z^{\max} d_z$. The right-hand side does not require any additional constraint evaluations because $c(z + \alpha_z^{\max} d_z)$ has been evaluated to measure the desirability of d. Since the factorization of the coefficient matrix has been performed during the computation of d, the total cost of the SOC step is one forward solve and one backsolve, plus one additional evaluation of the merit function (i.e., one additional function/constraint evaluation) to test the acceptance of the SOC point.

3.3 Slack Resets

Given the form (3.1) of the merit function,

$$\phi_\nu(z) = f(x) - \mu \sum_{i=1}^{m} \ln s_i + \nu \|c(z)\|,$$

and the fact that for fixed $\mu > 0$, the function $-\mu \ln s_i$ is a decreasing function of s_i, the algorithm resets certain slack components after each trial step in order to promote acceptance of the step by the merit function, while at the same time improving the satisfaction of the constraints. This reset is performed after a line search or trust region step is computed, and is as follows. Before the trial point $z^+ = z + \alpha_z d_z$ is tested for acceptance by the merit function, the slack components $s_i^+ = s_i + \alpha_z [d_s]_i$ for which the inequality $-g_i(x^+) > s_i^+$ is satisfied are reset to the value

$$s_i^+ = -g_i(x^+). \tag{3.13}$$

Thus (3.13) increases the slacks, thereby decreasing further the barrier term in (3.1) and improving the satisfaction of the constraints (2.1c), which decreases the penalty term $\|c(z)\|$. Hence, this reset may only cause the merit function to decrease further.

3.4 NLP Stopping Test

Ideally, the stopping test should be invariant under the scaling of the variables, the objective, and constraints. Achieving complete scale invariance is difficult, however, and in cases when certain quantities approach zero, it is not desirable. The following tests attempt to achieve a balance between practicality and scale invariance.

The stopping tolerances ϵ^{opt} and ϵ^{feas} are provided by the user. (In our tests they are both set to 10^{-6}.) The algorithm terminates if an iterate (x, s, λ) satisfies

$$\begin{aligned} \|\nabla f(x) + A_h(x)^T \lambda_h + A_g(x)^T \lambda_g\|_\infty &\leq \max\{1, \|\nabla f(x)\|_\infty\}\epsilon^{\text{opt}} && \text{(3.14a)} \\ \|S\lambda_g\|_\infty &\leq \max\{1, \|\nabla f(x)\|_\infty\}\epsilon^{\text{opt}} && \text{(3.14b)} \\ \|(h(x), g(x)^+)\|_\infty &\leq \max\{1, \|(h(x_0), g(x_0)^+)\|_\infty\}\epsilon^{\text{feas}}, && \text{(3.14c)} \end{aligned}$$

where $g(x)^+ = \max\{0, g(x)\}$ and x_0 is the starting point. When the maximum is not achieved by 1, the scaling factor in (3.14a) makes this test invariant to scalings in f, c and to linear changes of the variable x. The factor 1 is needed to safeguard the test when $\|\nabla f(x)\|$ is zero or nearly zero. The complementarity test (3.14b) is based on the fact that the scale of s is dependent on the scale of c and the magnitude of λ_g is proportional to $\|\nabla f\|/\|A\|$. Thus (3.14b) is invariant to the scaling of f and c. (A more complex scaling factor employing $\|A\|$ would make it invariant to linear changes in the variables, but we use the same scaling factors in (3.14a)-(3.14b) for simplicity.)

The scale factor for feasibility is difficult to choose. If the constraints are linear, then $\|(h(0), g(0)^+)\|_\infty$ is an appropriate normalization factor because it measures the magnitude of the right hand side vectors of all constraints. But since $h(0)$ or $g(0)$ may not be defined in some problems, we use the initial point, x_0.

3.5 Update of μ and Barrier Stopping Test

The sequence of barrier parameters $\{\mu_k\}$ must converge to zero, and should do so quickly if possible. Superlinear rules for decreasing μ have been studied in [11, 17, 22, 29], but they employ various parameters that can be difficult to select in practice. We use instead the following simple strategy for updating μ that has performed as well in our tests as have more complicated superlinear rules. If the most recent barrier problem was solved in less than three iterations, we set

$$\mu_{k+1} = \mu_k/100;$$

otherwise

$$\mu_{k+1} = \mu_k/5.$$

We now discuss the termination test for the barrier problem. Our experience is that the choice of this stopping test significantly affects the efficiency and robustness of interior methods. For the current value of μ, we choose tolerances $\epsilon_\mu^{\text{opt}}$ and $\epsilon_\mu^{\text{feas}}$ (defined below) and impose the following barrier stopping tests:

$$\begin{aligned} \|\nabla f(x) + A_h(x)^T \lambda_h + A_g(x)^T \lambda_g\|_\infty &\leq \max\{1, \|\nabla f(x)\|_\infty\}\epsilon_\mu^{\text{opt}} && \text{(3.15a)} \\ \|S\lambda_g - \mu e\|_\infty &\leq \max\{1, \|\nabla f(x)\|_\infty\}\epsilon_\mu^{\text{opt}} && \text{(3.15b)} \\ \|(h(x), g(x) + s)\|_\infty &\leq \max\{1, \|(h(x_0), g(x_0)^+\|_\infty\}\epsilon_\mu^{\text{feas}}. && \text{(3.15c)} \end{aligned}$$

Note that the scaling factors are identical to those used for the NLP stopping test (3.14a)-(3.14c) and that the left-hand sides of these two tests differ only in the use of the additional term $-\mu e$ in (3.15b) and the use of $g(x)+s$ rather than $g(x)^+$ in (3.15c).

As we shall see, the tolerances $\epsilon_\mu^{\rm opt}$ and $\epsilon_\mu^{\rm feas}$ will be chosen to be proportional to μ most of the time and thus become tighter as μ decreases. We want to avoid letting μ, $\epsilon_\mu^{\rm opt}$, and $\epsilon_\mu^{\rm feas}$ take unnecessarily small values because this can lead to significant roundoff errors. Therefore we determine the values of μ, $\epsilon_\mu^{\rm opt}$, and $\epsilon_\mu^{\rm feas}$ for which satisfaction of the barrier stopping test automatically implies satisfaction of the NLP stopping test.

Since the scaling factors and left-hand sides for (3.14a) and (3.15a) are identical, any point that satisfies (3.15a) automatically satisfies (3.14a) for all values $\epsilon_\mu^{\rm opt} \le \epsilon^{\rm opt}$. Also, from (3.15$b$) we have that for a given complementary pair $s_i[\lambda_g]_i$ that satisfies the barrier stopping test

$$s_i[\lambda_g]_i \le \mu + \max\{1, \|\nabla f(x)\|_\infty\}\epsilon_\mu^{\rm opt},$$

and in order to also satisfy the NLP stopping test (3.14b), it must satisfy

$$s_i[\lambda_g]_i \le \max\{1, \|\nabla f(x)\|_\infty\}\epsilon^{\rm opt}.$$

Therefore an iterate (x,s) that satisfies (3.15b) also satisfies (3.14b) as long as

$$\epsilon_\mu^{\rm opt} \le \epsilon^{\rm opt} - \mu/(\max\{1, \|\nabla f(x)\|_\infty\}). \tag{3.16}$$

Hence a point that satisfies the barrier KKT stopping conditions (3.15a),(3.15b) also satisfies the NLP KKT stopping conditions (3.14a),(3.14b) for all values of $\epsilon_\mu^{\rm opt}$ that satisfy

$$\epsilon_\mu^{\rm opt} \le \epsilon^{\rm opt} - \mu. \tag{3.17}$$

Note that this condition is only valid when the right-hand side of (3.17) is non-negative (i.e., $\mu \le \epsilon^{\rm opt}$).

For feasibility, since $s > 0$, we have that

$$\|g(x)^+\|_\infty \le \|g(x)+s\|_\infty.$$

Therefore from (3.14c) and (3.15c) it follows that for all

$$\epsilon_\mu^{\rm feas} \le \epsilon^{\rm feas} \tag{3.18}$$

the NLP feasibility test (3.14c) will be satisfied if the barrier feasibility test (3.15c) is satisfied.

Subject to the minimum values given by (3.17)-(3.18), we set $\epsilon_\mu^{\rm opt}$ and $\epsilon_\mu^{\rm feas}$ equal to $\theta\mu$, where θ is a fixed algorithm parameter (currently $\theta = 1$). The barrier stopping tolerances are then determined by the following formulas:

$$\epsilon_\mu^{\rm opt} = \max\{\theta\mu, \epsilon^{\rm opt} - \mu\} \tag{3.19}$$

$$\epsilon_\mu^{\rm feas} = \max\{\theta\mu, \epsilon^{\rm feas}\}. \tag{3.20}$$

This choice ensures that we do not oversolve the barrier subproblem, but it does not place any lower bound on the barrier parameter μ. Although an overly small value of μ

will not cause us to oversolve the barrier subproblem because of the limits on the barrier tolerances established above, we still want to prevent μ from becoming unnecessarily small because this can lead to failure of the iteration.

From (3.19) and (3.20), satisfaction of the barrier stopping test implies satisfaction of the NLP test if $\mu \le \min\{\epsilon^{\rm opt}/(1+\theta), \epsilon^{\rm feas}/\theta\}$. Currently $\theta = 1$; but to ensure that we do not restrict μ unduly, we enforce a minimum value of μ based on the NLP tolerance

$$\mu_{\min} = \frac{\min\{\epsilon^{\rm opt}, \epsilon^{\rm feas}\}}{100}. \tag{3.21}$$

3.6 Transition to and from Safeguarding Steps

If the line search step is not acceptable, the safeguarding trust region algorithm is invoked. It is desirable to provide this algorithm with a trust region radius Δ_k that reflects current problem information. This strategy is particularly important when two safeguarding trust region steps are separated by a long sequence of line search steps. In addition to preserving the global convergence properties of trust region methods, the size of Δ_k affects the backtracking line search, as explained in Section 3.2.

The algorithm uses the following strategy. If the most recent step was a successful line search step, d_{z_k}, we set

$$\Delta_{k+1} = 2\alpha_{z_k}\|d_{z_k}\|. \tag{3.22}$$

Otherwise, if the most recent step was either a trust region step or a rejected line search step, the trust region radius is updated according to standard trust region update rules. In our implementation we follow the update strategy given in [6]. Also, in order to try to avoid the repeated computation of unsuccessful line search steps, if a trust region iteration is rejected, subsequent iterations are computed with the trust region method until a successful step is obtained.

Numerical experience indicates that this simple strategy keeps the trust region information up to date and avoids oscillations in the size of Δ_k.

3.7 Solution of the Primal-Dual Equations

The primal-dual system (2.4) is solved by using the symmetric indefinite factorization implemented in the HSL library routine `MA27` [20]. This routine provides the number of negative eigenvalues, `neig`, of the primal-dual system. An important practical consideration is the choice of the pivot tolerance. We set it initially to 10^{-8} because such a small pivot tolerance can yield significant savings in computing time. If `MA27` returns a message of singularity, we increase it by a factor of 10 and compute a new factorization; this process is repeated if necessary until the maximum allowable pivot tolerance of 0.5 is reached. If the singularity message persists with the maximum pivot tolerance, the step is computed by using the latest factorization.

We also occasionally use iterative refinement. If the norm of the relative residual of the primal-dual system provided by the computed solution is greater than some threshold, we apply a maximum of five iterative refinement steps to try to decrease this residual and improve the solution.

3.8 Quasi-Newton Approximations

The algorithm includes several quasi-Newton options for problems in which second derivatives cannot be computed or when such a computation is too expensive. A quasi-Newton version of the primal-dual step is obtained by replacing $\nabla^2_{xx}\mathcal{L}$ in (2.6) by a quasi-Newton approximation B. Our algorithm implements dense BFGS and SR1 methods as well as a limited memory BFGS method.

In all these methods we initialize B to be the identity matrix. The correction pairs $(y, \Delta x)$ required by the quasi-Newton updating formula are obtained as follows. After computing a step from (z, λ) to (z^+, λ^+), we define

$$\begin{aligned} y &= \nabla_x \mathcal{L}(x^+, \lambda^+) - \nabla_x \mathcal{L}(x, \lambda^+) \\ \Delta x &= x^+ - x. \end{aligned}$$

The pair $(y, \Delta x)$ is then substituted into the standard definitions of the BFGS and SR1 updates to define B. In order to ensure that the BFGS method generates a positive definite matrix, the update is skipped if $y^T \Delta x \leq 0$. (A damping procedure is also an option.) SR1 updating is safeguarded by using standard rules; see, for example, [21, §8.2].

For large problems it is desirable to use limited memory updating to avoid the storage and manipulation of a dense $n \times n$ matrix. We have implemented a limited memory BFGS method using the compact representations described in [8]. Here B has the form

$$B = \xi I + NMN^T, \tag{3.23}$$

where $\xi > 0$ is a scaling factor, N is an $n \times 2p$ matrix, M is a $2p \times 2p$ symmetric and nonsingular matrix, and p denotes the number of correction pairs saved in the limited memory updating procedure. (Typical values of p are 5, 10, 20.) The matrices N and M are formed by using the vectors $\{\Delta x^k\}$ and $\{y^k\}$ accumulated in the last p iterations.

Since the limited memory matrix B is positive definite and A has full rank (otherwise we revert to the trust region step), the primal-dual matrix is nonsingular, and we can compute the solution to (2.4) by formally inverting the coefficient matrix. The computational cost is decreased by exploiting the low-rank structure of B. The inverse of the primal-dual matrix can be written as

$$\left(\begin{bmatrix} \xi I & 0 & A_h^T & A_g^T \\ 0 & S^{-1}\Lambda_g & 0 & I \\ A_h & 0 & 0 & 0 \\ A_g & I & 0 & 0 \end{bmatrix} + \begin{bmatrix} N \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} MN^T & 0 & 0 & 0 \end{bmatrix} \right)^{-1} \equiv [C + UV^T]^{-1}.$$

Applying the Sherman-Morrison-Woodbury formula, we can write the right-hand side as

$$C^{-1} - C^{-1}U\left(I + V^T C^{-1} U\right)^{-1} V^T C^{-1}. \tag{3.24}$$

The computation of the primal-dual step (2.4) therefore requires the solution of systems of the form $Cz = b$, which is done by means of the HSL routine `MA27`. The matrix inside the parentheses in (3.24) is small (of dimension $2p \times 2p$) and can be factored at negligible cost.

4 Numerical Results

The algorithm described in the previous sections has been implemented in the KNITRO package. We refer to the new algorithm as Knitro-Direct because in practice the majority of the iterations are obtained by factoring the primal-dual system. We compare it with the trust region algorithm in KNITRO 3.0, henceforth called Knitro-CG. We use the CUTEr collection [3, 18] as of June 5, 2003, from which 968 problems have been retained; the remaining problems have been discarded because they require too much memory.

All tests were performed on a 2.8 GHz Pentium Xeon, with 3 Gb of memory running Red Hat Linux. KNITRO 3.0 is written in C and Fortran 77 and was compiled by using the gcc and g77 compilers with the "-O" compilation flag and was run in double precision. Limits of 15 minutes of CPU time and 3000 outer iterations were imposed for each problem; if one of these limits was reached, the algorithm was considered to have failed. The stopping tolerances in (3.14a)-(3.14c) were set as $\epsilon^{\text{opt}} = \epsilon^{\text{feas}} = 10^{-6}$. For details on the convergence criteria used in KNITRO 3.0, see [28].

We report results using the logarithmic performance profiles proposed by Dolan and Moré [10]. Let $t_{p,s}$ denote the time to solve problem p by solver s. We define the ratios

$$r_{p,s} = \frac{t_{p,s}}{t_p^*}, \tag{4.25}$$

where t_p^* is the lowest time required by any code to solve problem p. If a code does not solve a problem, the ratio $r_{p,s}$ is assigned a large number, M. Then, the logarithmic performance profile for each code s, is

$$\pi_s(\tau) = \frac{\text{no. of problems s.t.} \log_2(r_{p,s}) \leq \tau}{\text{total no. of problems}}, \qquad \tau \geq 0. \tag{4.26}$$

This performance profile will also be used to analyze the number of function evaluations required by the four codes.

First, we compare in Figure 1 the relative performance of Knitro-Direct and Knitro-CG in terms of number of function/constraint evaluations, on the 968 problems from CUTEr. In this and the figures that follow, we plot π_s as a function of τ. Observe that the two algorithms are remarkably similar in terms of robustness and efficiency, and a profile based on iterations is almost identical to that given in Figure 1. We note that, even though these two algorithms are quite different in the way they generate steps, they share many other features, including common NLP and barrier stop tests, merit function, second-order correction, and barrier parameter update strategy.

Next, we compare computing times. Since timing results are uninteresting and can be unreliable on small problems that are solved very quickly, we consider only those problems for which $n + \bar{m} \geq 1000$, where $\bar{m}$ is the number of equality and general inequalities (excluding bounds). Figure 2 gives the performance profile for all CUTEr problems with this size restriction.

One might expect that Knitro-Direct will require less computing time than Knitro-CG on problems in which the Hessian of the Lagrangian of the nonlinear program is ill-conditioned

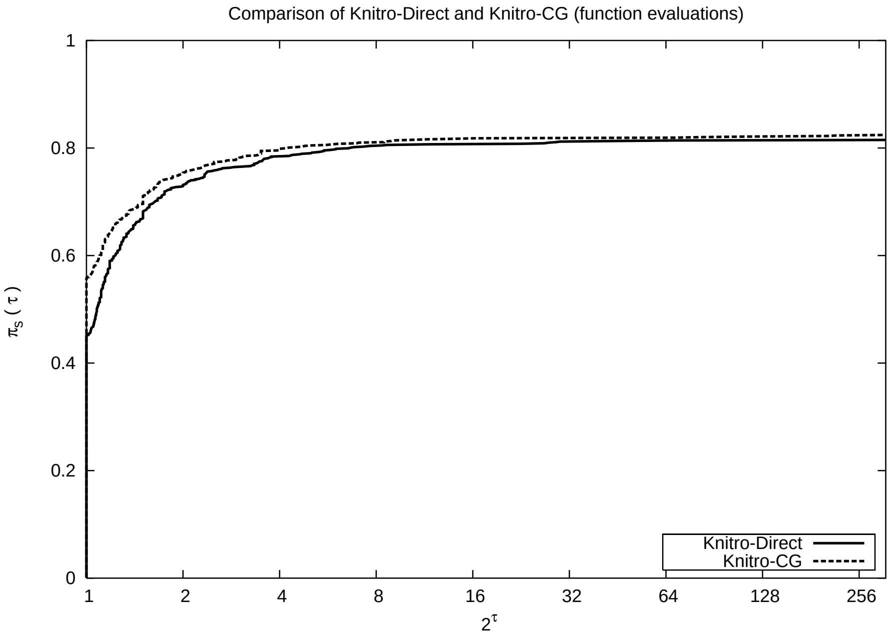

Figure 1: Comparing number of function evaluations for Knitro-Direct and Knitro-CG on 968 CUTEr problems.

(causing a very large number of CG iterations) and where factoring the KKT matrix is not excessively expensive. A detailed examination of the results shows that this is the case, but we note that the problems in which Knitro-Direct has a clear advantage in speed are also characterized by the fact that negative curvature was rarely encountered.

For example, when considering the 29 problems in which Knitro-Direct was at least 10 times faster than Knitro-CG, typically 90% of Knitro-CG's time was taken by the CG iteration (i.e., the Hessian-vector products and backsolves involving the projection matrix), and only in one of those problems did Knitro-Direct report negative curvature. On the other hand, in the runs in which Knitro-Direct required much more computing time than Knitro-CG, the factorization of the primal-dual matrix was very expensive (requiring typically 85% of the total time or more) or negative curvature was encountered frequently.

4.1 Transition Between Algorithms and Negative Curvature

Table 1 provides statistics on the transition to trust region steps. It is based on the 788 CUTEr problems in which Knitro-Direct reported finding an optimal solution. The table gives the percentage of iterations in which the trust region step was invoked, and then gives a breakdown of this number in terms of the three factors that can cause this in Algorithm 2.1: (i) negative curvature was encountered, that is, `neig` $> l + m$; (ii) the line search did not succeed in reducing the merit function after 3 backtracking steps (backtrack), or (iii) the primal-dual step was too long in the sense that $\min\{\alpha_z^{\max}, \alpha_\lambda^{\max}\} \le \delta$ (cut-back). The first

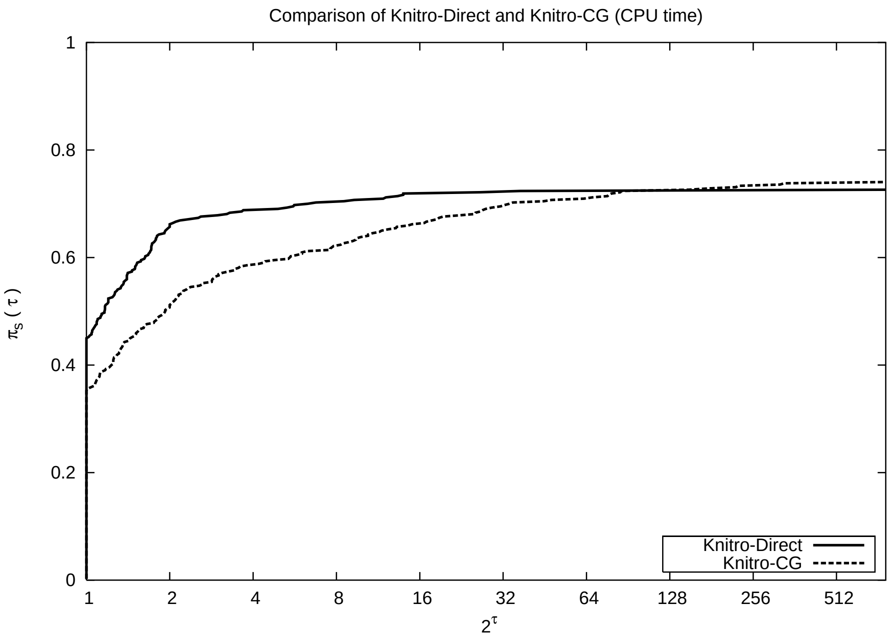

Figure 2: CPU time for Knitro-Direct and Knitro-CG on 420 CUTEr problems for which $n + \bar{m} \geq 1000$.

Prob. Type	# probs	TR invoked	neg curv	backtrack	cut-back
All	788	22.9%	15.9%	6.0%	1.0%
Inequality	481	22.3%	13.8%	7.1%	1.4%

Table 1: Frequency and reason for invoking trust region step.

row of this table looks at the whole set of 788 problems, while the second row looks only at the 481 problems from this set that have at least one inequality constraint or bound (such that the cut-back procedure for the primal-dual step is relevant). As Table 1 shows, the great majority of steps taken in the Knitro-Direct algorithm are direct steps. Moreover, the iteration reverts to trust region steps mainly because negative curvature is detected, and not because of the occurrence of very short steplengths.

4.2 Quasi-Newton Options

We first analyzed the performance of the three quasi-Newton options in Knitro-Direct on all small and medium-size problems in CUTEr, that is problems with $n + \bar{m} < 1000$. This size restriction is necessary because Knitro-Direct-BFGS and Knitro-Direct-SR1 implement dense quasi-Newton approximations to the Hessian of the Lagrangian. Knitro-Direct-LM refers to the limited memory BFGS option storing 20 correction pairs (i.e., $p = 20$ in the notation of §3.8). Figure 3 compares these three options in terms of function/constraint evaluations.

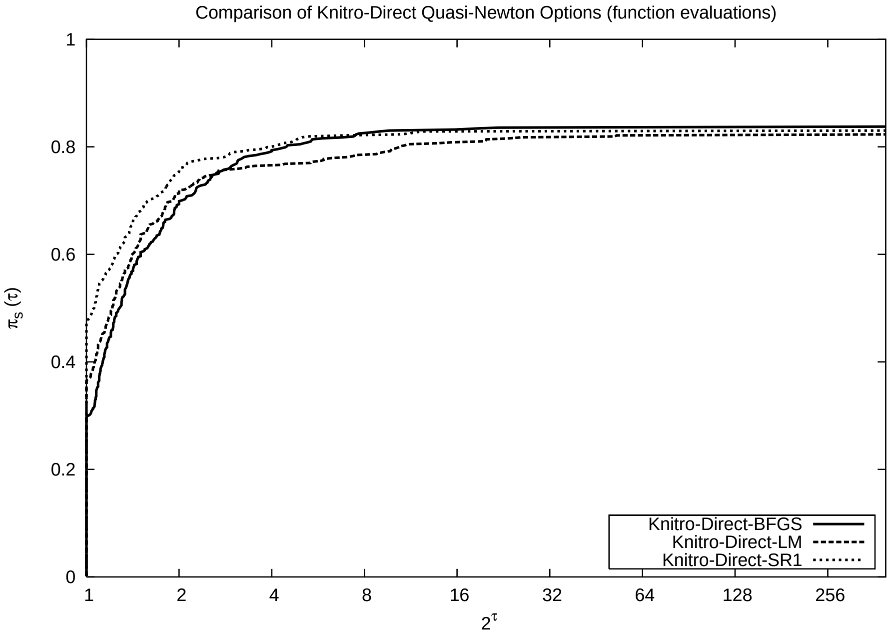

Figure 3: Number of function evaluations for 3 quasi-Newton versions of Knitro-Direct on 548 CUTEr problems for which $n + \bar{m} < 1000$.

In these tests the SR1 method performs slightly better than BFGS. More significantly, the limited memory BFGS method is very similar in performance to its dense counterpart. This, and the fact that it is applicable to large problems, suggests that it can be used as the default quasi-Newton option in Knitro-Direct.

Next, we compare in Figure 4 the performance, in terms of function evaluations, of the following 4 methods: (i) Knitro-Direct with exact Hessian (labeled Knitro-Direct); (ii) Knitro-Direct with limited memory BFGS updating (Knitro-Direct-LM); (iii) Knitro-CG using limited memory BFGS (Knitro-CG-LM); and (iv) Knitro-CG with an option [28] in which Hessian-vector products are computed by finite differences of gradients (Knitro-CG-FD). As before, in the limited memory methods we stored 20 correction pairs. Here we use the complete test set of 968 problems of all sizes.

As expected, the quasi-Newton versions are less robust and efficient than the second derivative options, but their performance is acceptable. Of 968 problems, Knitro-Direct with second derivatives solved 788 problems, whereas Knitro-Direct with limited memory BFGS solved 693. We note that the finite-difference option is only slightly less efficient, in terms of function evaluations, than the methods using exact second derivatives. However, if gradient evaluations are expensive, this option may be inefficient in terms of CPU time.

Acknowledgments. We thank Guanghui Liu and Marcelo Marazzi for their work on an

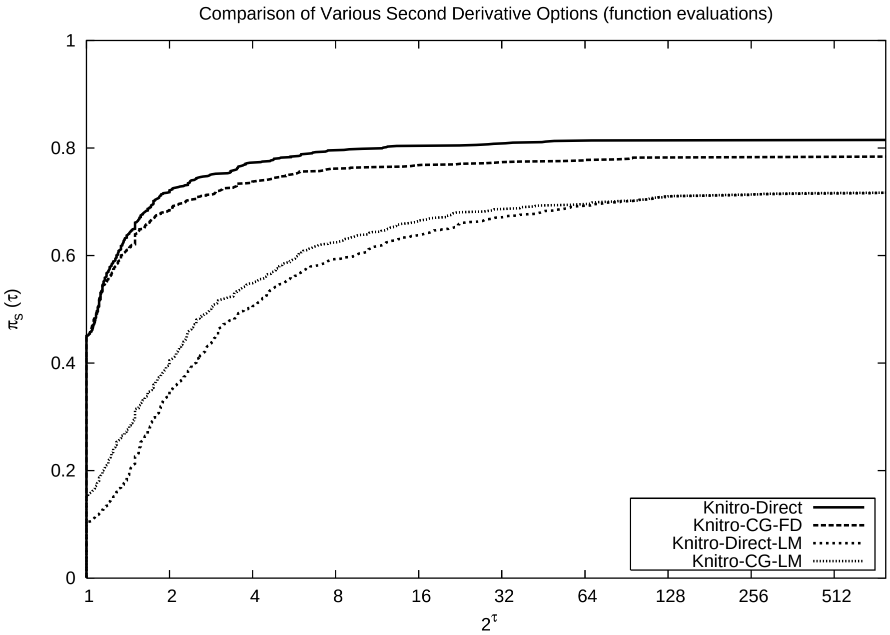

Figure 4: Number of function evaluations for options using exact Hessian, limited memory quasi-Newton, and finite differences on 968 CUTEr problems.

early version of this software. We also thank Richard Byrd for several useful conversations on the design of this algorithm.

References

[1] E. D. Andersen, J. Gondzio, C. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior Point Methods in Mathematical Programming*, pages 189–252, Dordrecht, The Netherlands, 1996. Kluwer Academic Publishers.

[2] J. Betts, S. K. Eldersveld, P. D. Frank, and J. G. Lewis. An interior-point nonlinear programming algorithm for large scale optimization. Technical report MCT TECH-003, Mathematics and Computing Technology, The Boeing Company, P.O. Box 3707, Seattle WA 98124-2207, 2000.

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

[4] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655, 1971.

[5] R. H. Byrd, J.-Ch. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000.

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

[7] R. H. Byrd, M. Marazzi, and J. Nocedal. On the convergence of Newton iterations to non-stationary points. *Mathematical Programming, Series A*, 99:127–148, 2004.

[8] R. H. Byrd, J. Nocedal, and R. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming*, 49(3):285–323, 1991.

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

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

[11] J.-P. Dussault. Numerical stability and efficiency of penalty algorithms. *SIAM Journal on Numerical Analysis*, 32(1):296–317, 1995.

[12] A. S. El-Bakry, R. A. Tapia, T. Tsuchiya, and Y. Zhang. On the formulation and theory of the Newton interior-point method for nonlinear programming. *Journal of Optimization Theory and Applications*, 89(3):507–541, June 1996.

[13] M. El-Hallabi. A hybrid algorithm for nonlinear equality constrained optimization problems: global and local convergence theory. Technical Report TR4-99, Mathematics and Computer Science Department, Institut National des Postes et Télécommunications, Rabat, Morocco, 1999.

[14] R. Fletcher. *Practical Methods of Optimization*. J. Wiley and Sons, Chichester, England, second edition, 1987.

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

[16] A. Forsgren and P. E. Gill. Primal-dual interior methods for nonconvex nonlinear programming. *SIAM Journal on Optimization*, 8(4):1132–1152, 1998.

[17] N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Superlinear convergence of primal-dual interior-point algorithms for nonlinear programming. *SIAM Journal on Optimization*, 11(4):974–1002, 2001.

[18] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr and sifdec: A Constrained and Unconstrained Testing Environment, revisited. *ACM Trans. Math. Softw.*, 29(4):373–394, 2003.

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

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

[21] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research. Springer, 1999.

[22] F. Potra. Q-superlinear convergence of the iterates in primal-dual interior-point methods. *Mathematical Programming B*, 91(1):99–116, 2001.

[23] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[24] A. L. Tits, A. Wächter, S. Bakhtiari, T. J. Urban, and C. T. Lawrence. A primal-dual interior-point method for nonlinear programming with strong global and local convergence properties. *SIAM Journal on Optimization*, 14(1):173–199, 2003.

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

[26] A. Wächter and L. T. Biegler. Failure of global convergence for a class of interior point methods for nonlinear programming. *Mathematical Programming*, 88(3):565–574, 2000.

[27] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Technical Report RC 23149, IBM T.J. Watson Research Center, Yorktown Heights, NY, March 2004.

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

[29] H. Yabe and H. Yamashita. Q-superlinear convergence of primal-dual interior point quasi-Newton methods for constrained optimization. *Journal of the Operations Research Society of Japan*, 40(3):415–436, 1997.

[30] H. Yamashita. A globally convergent primal-dual interior-point method for constrained optimization. Technical report, Mathematical System Institute, Inc., Tokyo, Japan, May 1992. Revised March 1994.