Math. Program., Ser. A 106, 25–57 (2006)

Digital Object Identifier (DOI) 10.1007/s10107-004-0559-y

Andreas Wächter · Lorenz T. Biegler

On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming

Received: March 12, 2004 / Accepted: September 2, 2004
Published online: April 28, 2005 –

Abstract. We present a primal-dual interior-point algorithm with a filter line-search method for nonlinear programming. Local and global convergence properties of this method were analyzed in previous work. Here we provide a comprehensive description of the algorithm, including the feasibility restoration phase for the filter method, second-order corrections, and inertia correction of the KKT matrix. Heuristics are also considered that allow faster performance. This method has been implemented in the IPOPT code, which we demonstrate in a detailed numerical study based on 954 problems from the CUTEr test set. An evaluation is made of several line-search options, and a comparison is provided with two state-of-the-art interior-point codes for nonlinear programming.

Key words. Nonlinear programming – Nonconvex constrained optimization – Filter method – Line search – Interior-point method – Barrier method

1. Introduction

Growing interest in efficient optimization methods has led to the development of *interior-point* or *barrier* methods for large-scale nonlinear programming. In particular, these methods provide an attractive alternative to active set strategies in handling problems with large numbers of inequality constraints. Over the past 15 years, there has also been a better understanding of the convergence properties of interior-point methods [16] and efficient algorithms have been developed with desirable global and local convergence properties.

To allow convergence from poor starting points, interior-point methods, in both trust region and line-search frameworks, have been developed that use exact penalty merit functions to enforce progress toward the solution [2, 21, 29]. On the other hand, Fletcher and Leyffer [14] recently proposed filter methods, offering an alternative to merit functions, as a tool to guarantee global convergence in algorithms for nonlinear programming. The underlying concept is that trial points are accepted if they improve the objective function *or* improve the constraint violation instead of a combination of those two measures defined by a merit function.

More recently, this filter approach has been adapted to barrier methods in a number of ways. M. Ulbrich, S. Ulbrich, and Vicente [22] consider a trust region filter method

Andreas Wächter: IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY, 10598, USA.
e-mail: andreasw@watson.ibm.com

Lorenz T. Biegler: Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213, USA.
e-mail: lb01@andrew.cmu.edu

Mathematical Subject Classification (2000): 49M37, 65K05, 90C30, 90C51

that bases the acceptance of trial steps on the norm of the optimality conditions. Also, Benson, Shanno, and Vanderbei [1] proposed several heuristics based on the idea of filter methods, for which improved efficiency is reported compared to their previous merit function approach, although no convergence analysis is given. Finally, global convergence of an interior-point algorithm with a filter line search is analyzed in [26]. The assumptions made for the analysis of the interior-point method in [26] are less restrictive than those made for previously proposed line-search interior-point methods for nonlinear programming (e.g., [10, 29]).

A number of interior-point methods have been implemented in robust software codes (such as [3, 23]), and numerical tests have shown them to be efficient and robust in practice. In this paper we describe the detailed development of a primal-dual interior-point algorithm with a filter line-search, based on the analysis in [26]. We consider a primal-dual barrier method to solve nonlinear optimization problems of the form

$$\min_{x\in\mathbb{R}^n} \quad f(x) \tag{1a}$$

$$\text{s.t.} \quad c(x) = 0 \tag{1b}$$

$$x_L \le x \le x_U, \tag{1c}$$

where $x_L \in [-\infty, \infty)^n$ and $x_U \in (-\infty, \infty]^n$, with $x_L^{(i)} \le x_U^{(i)}$, are the lower and upper bounds on the variables x. The objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and the equality constraints $c : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, with $m \le n$, are assumed to be twice continuously differentiable. Problems with general nonlinear inequality constraints, "$d(x) \le 0$," can be reformulated in the above form by introducing slack variables.

The paper is organized as follows. Section 2 presents the overall algorithm, including the step computation, the filter line-search procedure, and a second-order correction. In Section 3, we describe some aspects of the algorithm, and its implementation, in more detail, including the restoration phase for the filter procedure, as well as several heuristics to improve the performance of the overall method. Section 4 presents numerical results of our implementation, called IPOPT, for the CUTEr test set [18], including a comparison of the filter method with a penalty function approach, and a comparison with two state-of-the-art nonlinear optimization codes, KNITRO [3, 28] and LOQO [23].

1.1. Notation

The i-th component of a vector $v \in \mathbb{R}^n$ is written as $v^{(i)}$. Norms $\|\cdot\|$ denote a fixed vector norm and its compatible matrix norm unless explicitly noted. We further introduce the notation $X := \mathrm{diag}(x)$ for a vector x (similarly $Z := \mathrm{diag}(z)$, etc.), and e stands for the vector of all ones for appropriate dimension.

Finally, we will refer to ϵ_{mach} as the machine precision for the finite arithmetic. For the computer and implementation used for our numerical experiments, it is $\epsilon_{\mathrm{mach}} \approx 10^{-16}$. The algorithm presented in this paper has parameters, which have to be given values for a practical implementation. Except where explicitly noted, these parameters do not depend on the details of the finite arithmetic.

2. The Basic Algorithm

The following sections motivate the proposed algorithm, which is formally summarized in Section 2.5.

2.1. The Primal-Dual Barrier Approach

To simplify notation, we first describe the method for the problem formulation

$$\min_{x\in\mathbb{R}^n} \quad f(x) \tag{2a}$$
$$\text{s.t.} \quad c(x) = 0 \tag{2b}$$
$$x \geq 0. \tag{2c}$$

The changes necessary to handle the general case (1) will be briefly outlined in Section 3.4.

As a barrier method, like the methods discussed in [2, 8, 11, 29], the proposed algorithm computes (approximate) solutions for a sequence of barrier problems

$$\min_{x\in\mathbb{R}^n} \quad \varphi_\mu(x) := f(x) - \mu \sum_{i=1}^{n} \ln(x^{(i)}) \tag{3a}$$
$$\text{s.t.} \quad c(x) = 0 \tag{3b}$$

for a decreasing sequence of barrier parameters μ converging to zero. Equivalently, this can be interpreted as applying a homotopy method to the primal-dual equations,

$$\nabla f(x) + \nabla c(x)\lambda - z = 0 \tag{4a}$$
$$c(x) = 0 \tag{4b}$$
$$XZe - \mu e = 0, \tag{4c}$$

with the homotopy parameter μ, which is driven to zero (see e.g., [4, 17]). Here, $\lambda \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$ correspond to the Lagrangian multipliers for the equality constraints (2b) and the bound constraints (2c), respectively. Note, that the equations (4) for $\mu = 0$ together with "$x, z \geq 0$" are the Karush-Kuhn-Tucker (KKT) conditions for the original problem (2). Those are the first order optimality conditions for (2) if constraint qualifications are satisfied [7].

The presented method computes an approximate solution to the barrier problem (3) for a fixed value of μ, then decreases the barrier parameter, and continues the solution of the next barrier problem from the approximate solution of the previous one.

Using the individual parts of the primal-dual equations (4), we define the optimality error for the barrier problem as

$$E_\mu(x, \lambda, z) := \max\left\{ \frac{\|\nabla f(x) + \nabla c(x)\lambda - z\|_\infty}{s_d}, \|c(x)\|_\infty, \frac{\|XZe - \mu e\|_\infty}{s_c} \right\} \tag{5}$$

with scaling parameters $s_d, s_c \geq 1$ defined below. By $E_0(x, \lambda, z)$ we denote (5) with $\mu = 0$; this measures the optimality error for the original problem (2). The overall algorithm terminates if an approximate solution $(\tilde{x}_*, \tilde{\lambda}_*, \tilde{z}_*)$ (including multiplier estimates) satisfying

$$E_0(\tilde{x}_*, \tilde{\lambda}_*, \tilde{z}_*) \leq \epsilon_{\text{tol}} \tag{6}$$

is found, where $\epsilon_{\text{tol}} > 0$ is the user provided error tolerance.

Even if the original problem is well scaled (see also Section 3.8 on automatic scaling of the objective and constraint functions), the multipliers λ and z might become very large, for example, when the gradients of the active constraints are (nearly) linearly dependent at a solution of (2). In this case, the algorithm might encounter numerical difficulties satisfying the unscaled primal-dual equations (4) to a tight tolerance. In order to adapt the termination criteria to handle such circumstances, we choose the scaling factors

$$s_d = \max\left\{s_{\max}, \frac{\|\lambda\|_1 + \|z\|_1}{(m+n)}\right\} / s_{\max} \qquad s_c = \max\left\{s_{\max}, \frac{\|z\|_1}{n}\right\} / s_{\max}$$

in (5). In this way, a component of the optimality error is scaled, whenever the average value of the multipliers becomes larger than a fixed number $s_{\max} \geq 1$ ($s_{\max} = 100$ in our implementation). Also note, in the case that the multipliers diverge, $E_0(x, \lambda, z)$ can only become small, if a Fritz-John point for (2) is approached, or if the primal variables diverge as well.

In order to achieve fast local convergence (to a local solution of (2) satisfying the strong second-order sufficient optimality conditions), we follow the approach proposed by Byrd, Liu, and Nocedal [4, Strategy 2], which is proven to give rise to superlinear convergence under standard second-order sufficient conditions. Denoting with j the iteration counter for the "outer loop," we require that the approximate solution $(\tilde{x}_{*,j+1}, \tilde{\lambda}_{*,j+1}, \tilde{z}_{*,j+1})$ of the barrier problem (3), for a given value of μ_j, satisfies the tolerance

$$E_{\mu_j}(\tilde{x}_{*,j+1}, \tilde{\lambda}_{*,j+1}, \tilde{z}_{*,j+1}) \leq \kappa_\epsilon \mu_j$$

for a constant $\kappa_\epsilon > 0$, before the algorithm continues with the solution of the next barrier problem. The new barrier parameter is obtained from

$$\mu_{j+1} = \max\left\{\frac{\epsilon_{\text{tol}}}{10}, \min\left\{\kappa_\mu \mu_j,\ \mu_j^{\theta_\mu}\right\}\right\}, \tag{7}$$

with constants $\kappa_\mu \in (0, 1)$ and $\theta_\mu \in (1, 2)$. In this way, the barrier parameter is eventually decreased at a superlinear rate. On the other hand, the update rule (7) does not allow μ to become smaller than necessary given the desired tolerance ϵ_{tol}, thus avoiding numerical difficulties at the end of the optimization procedure.

For later reference, we also choose a "fraction-to-the-boundary" parameter

$$\tau_j = \max\{\tau_{\min}, 1 - \mu_j\} \tag{8}$$

where $\tau_{\min} \in (0, 1)$ is its minimum value.

2.2. Solution of the Barrier Problem

In order to solve the barrier problem (3) for a given fixed value μ_j of the barrier parameter, a damped Newton's method is applied to the primal-dual equations (4). Here, we use k to denote the iteration counter for the "inner loop." Given an iterate (x_k, λ_k, z_k) with $x_k, z_k > 0$, search directions $(d_k^x, d_k^\lambda, d_k^z)$ are obtained from the linearization of (4) at (x_k, λ_k, z_k), namely

$$\begin{bmatrix} W_k & A_k & -I \\ A_k^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \\ d_k^z \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k \lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu_j e \end{pmatrix}. \tag{9}$$

Here $A_k := \nabla c(x_k)$, and W_k denotes the Hessian $\nabla^2_{xx} \mathcal{L}(x_k, \lambda_k, z_k)$ of the Lagrangian function (for the original problem (2)),

$$\mathcal{L}(x, \lambda, z) := f(x) + c(x)^T \lambda - z. \tag{10}$$

Instead of solving the nonsymmetric linear system (9) directly, the proposed method computes the solution equivalently by first solving the smaller, symmetric linear system

$$\begin{bmatrix} W_k + \Sigma_k & A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_{\mu_j}(x_k) + A_k \lambda_k \\ c(x_k) \end{pmatrix}, \tag{11}$$

with $\Sigma_k := X_k^{-1} Z_k$, derived from (9) by eliminating the last block row. The vector d_k^z is then obtained from

$$d_k^z = \mu_j X_k^{-1} e - z_k - \Sigma_k d_k^x. \tag{12}$$

As is common for most line-search methods, we have to ensure that the matrix in the top-left block in the matrix in (11), projected onto the null space of the constraint Jacobian A_k^T, is positive definite. This is necessary to guarantee certain descent properties for the filter line-search procedure below [26]. Also, if A_k does not have full rank, the iteration matrix in (11) is singular, and a solution of (11) might not exist. Therefore, it might be necessary to modify the iteration matrix. In our implementation, we solve the linear system

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & A_k \\ A_k^T & -\delta_c I \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_{\mu_j}(x_k) + A_k \lambda_k \\ c(x_k) \end{pmatrix}, \tag{13}$$

for some $\delta_w, \delta_c \geq 0$. The choice of the scalars δ_w and δ_c for each iteration is discussed below in Section 3.1.

Having computed search directions from (13) and (12), step sizes $\alpha_k, \alpha_k^z \in (0, 1]$ have to be determined in order to obtain the next iterate as

$$x_{k+1} := x_k + \alpha_k d_k^x \tag{14a}$$

$$\lambda_{k+1} := \lambda_k + \alpha_k d_k^\lambda \tag{14b}$$

$$z_{k+1} := z_k + \alpha_k^z d_k^z. \tag{14c}$$

Note that we allow a different step size in the z variables from that of the other variables. In our experience, this is more efficient since it does not unnecessarily restrict the steps.

Since x and z are both positive at an optimal solution of the barrier problem (3), this property is maintained for all iterates. It is attained using the *fraction-to-the-boundary* rule

$$\alpha_k^{\max} := \max\left\{\alpha \in (0,1] : x_k + \alpha d_k^x \geq (1-\tau_j)x_k\right\} \tag{15a}$$

$$\alpha_k^{z} := \max\left\{\alpha \in (0,1] : z_k + \alpha d_k^z \geq (1-\tau_j)z_k\right\} \tag{15b}$$

where the parameter $\tau_j \in (0,1)$ is defined in (8). Note that α_k^z is the actual step size used in (14c). In order to ensure global convergence, the step size $\alpha_k \in (0, \alpha_k^{\max}]$ for the remaining variables is determined by a backtracking line-search procedure exploring a decreasing sequence of trial step sizes $\alpha_{k,l} = 2^{-l}\alpha_k^{\max}$ (with $l = 0, 1, 2, \ldots$). We use a line-search variant of Fletcher and Leyffer's filter method [14], which we present and analyze in [26]. In particular, in [26] we prove that this procedure is globally convergent under appropriate (mild) assumptions.

Before reviewing this procedure in the next section, we briefly note that a requirement for the convergence proof in [26] is that the "primal-dual barrier term Hessian" Σ_k does not deviate arbitrarily much from the "primal Hessian" $\mu_j X_k^{-2}$. We ensure this by resetting

$$z_{k+1}^{(i)} \leftarrow \max\left\{\min\left\{z_{k+1}^{(i)}, \frac{\kappa_\Sigma\ \mu_j}{x_{k+1}^{(i)}}\right\}, \frac{\mu_j}{\kappa_\Sigma\ x_{k+1}^{(i)}}\right\}, \qquad i = 1, \ldots, n, \tag{16}$$

for some fixed $\kappa_\Sigma \geq 1$ after each step. This guarantees that each component $\sigma_{k+1}^{(i)}$ of Σ_{k+1} is in the interval

$$\sigma_{k+1}^{(i)} \in \left[\frac{1}{\kappa_\Sigma}\mu_j/(x_k^{(i)})^2, \kappa_\Sigma\mu_j/(x_k^{(i)})^2\right]. \tag{17}$$

Such safeguards are common for the global convergence proof of primal-dual methods for NLP (see e.g., [8, 30]), and do not interfere with the primal-dual spirit of the method in terms of local convergence, when the parameter κ_Σ is chosen sufficiently large. In our implementation, $\kappa_\Sigma = 10^{10}$.

2.3. A Line-Search Filter Method

Filter methods were originally proposed by Fletcher and Leyffer [14]. In the context of solving the barrier problem (3) for μ_j, the basic idea behind this approach is to interpret (3) as a bi-objective optimization problem with the two goals of minimizing the objective function $\varphi_{\mu_j}(x)$ and the constraint violation $\theta(x) := \|c(x)\|$ (with a certain emphasis on the latter quantity). Following this paradigm, we might consider a trial point $x_k(\alpha_{k,l}) := x_k + \alpha_{k,l}d_k^x$ during the backtracking line search to be acceptable, if it leads to sufficient progress toward either goal compared to the current iterate, i.e., if

$$\theta(x_k(\alpha_{k,l})) \leq (1-\gamma_\theta)\theta(x_k) \tag{18a}$$

$$\text{or} \qquad \varphi_{\mu_j}(x_k(\alpha_{k,l})) \leq \varphi_{\mu_j}(x_k) - \gamma_\varphi\theta(x_k) \tag{18b}$$

holds for fixed constants $\gamma_\theta, \gamma_\varphi \in (0, 1)$. However, the above criterion is replaced by requiring sufficient progress in the barrier objective function, whenever for the current iterate we have $\theta(x_k) \leq \theta^{\min}$, for some constant $\theta^{\min} \in (0, \infty]$, and the following "switching condition"

$$\nabla\varphi_{\mu_j}(x_k)^T d_k^x < 0 \qquad \text{and} \qquad \alpha_{k,l}[-\nabla\varphi_{\mu_j}(x_k)^T d_k^x]^{s_\varphi} > \delta\,[\theta(x_k)]^{s_\theta}\,, \tag{19}$$

with constants $\delta > 0, s_\theta > 1, s_\varphi \geq 1$ holds. If $\theta(x_k) \leq \theta^{\min}$ and (19) is true for the current step size $\alpha_{k,l}$, the trial point has to satisfy the Armijo condition

$$\varphi_{\mu_j}(x_k(\alpha_{k,l})) \leq \varphi_{\mu_j}(x_k) + \eta_\varphi \alpha_{k,l} \nabla\varphi_{\mu_j}(x_k)^T d_k^x, \tag{20}$$

instead of (18), in order to be acceptable. Here, $\eta_\varphi \in (0, \frac{1}{2})$ is a constant. If the projection of the top-left matrix in (13) onto the null space of A_k^T is uniformly positive definite, it can be shown that condition (19) becomes true if a feasible, but non-optimal point is approached. Enforcing decrease in the objective function by (20) then prevents the method from converging to such a point. In accordance with previous publications on filter methods (e.g. [13, 15]) we may call a trial step size $\alpha_{k,l}$ for which (19) holds, a "φ-step size."

The algorithm also maintains a "filter," a set $\mathcal{F}_k \subseteq \{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq 0\}$ for each iteration k. The filter $\mathcal{F}_k$ contains those combinations of constraint violation values θ and the objective function values φ, that are "prohibited" for a successful trial point in iteration k: During the line search, a trial point $x_k(\alpha_{k,l})$ is rejected, if $(\theta(x_k(\alpha_{k,l})), \varphi_{\mu_j}(x_k(\alpha_{k,l}))) \in \mathcal{F}_k$. We then say, that the trial point is not acceptable to the current filter. At the beginning of the optimization, the filter is initialized to

$$\mathcal{F}_0 := \{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq \theta^{\max}\} \tag{21}$$

for some $\theta^{\max}$, so that the algorithm will never allow trial points to be accepted that have a constraint violation larger than $\theta^{\max}$. Later, the filter is augmented, using the update formula

$$\mathcal{F}_{k+1} := \mathcal{F}_k \cup \left\{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq (1 - \gamma_\theta)\theta(x_k) \quad \text{and} \quad \varphi \geq \varphi_{\mu_j}(x_k) - \gamma_\varphi \theta(x_k)\right\}, \tag{22}$$

after every iteration, in which the accepted trial step size does not satisfy the switching condition (19), or in which the Armijo condition (20) does not hold. This ensures that the iterates cannot return to the neighborhood of x_k. On the other hand, if both (19) and (20) hold for the accepted step size, the filter remains unchanged.

Overall, this procedure ensures that the algorithm cannot cycle, for example between two points that alternatingly decrease the constraint violation and the barrier objective function.

Finally, in some cases it is not possible to find a trial step size $\alpha_{k,l}$ that satisfies the above criteria. We approximate a minimum desired step size using linear models of the involved functions. For this, we define

$$\alpha_k^{\min} := \gamma_\alpha \cdot \begin{cases} \min\left\{\gamma_\theta, \frac{\gamma_\varphi \theta(x_k)}{-\nabla\varphi_{\mu_j}(x_k)^T d_k^x}, \frac{\delta[\theta(x_k)]^{s_\theta}}{[-\nabla\varphi_{\mu_j}(x_k)^T d_k^x]^{s_\varphi}}\right\} \\ \qquad \text{if } \nabla\varphi_{\mu_j}(x_k)^T d_k^x < 0 \text{ and } \theta(x_k) \le \theta^{\min} \\ \min\left\{\gamma_\theta, \frac{\gamma_\varphi \theta(x_k)}{-\nabla\varphi_{\mu_j}(x_k)^T d_k^x}\right\} \\ \qquad \text{if } \nabla\varphi_{\mu_j}(x_k)^T d_k^x < 0 \text{ and } \theta(x_k) > \theta^{\min} \\ \gamma_\theta \\ \qquad \text{otherwise,} \end{cases} \tag{23}$$

with a "safety factor" $\gamma_\alpha \in (0, 1]$. If the backtracking line search encounters a trial step size with $\alpha_{k,l} \le \alpha_k^{\min}$, the algorithm reverts to a *feasibility restoration phase*. Here, the algorithm tries to find a new iterate $x_{k+1} > 0$ which is acceptable to the current filter and for which (18) holds, by reducing the constraint violation with some iterative method.

Note that the restoration phase algorithm might not be able to produce a new iterate for the filter line-search method, for example, when the problem is infeasible. In this case, a suitable restoration phase algorithm should converge to a local minimizer (or at least a stationary point) for the constraint violation, indicating to the user that the problem seems (at least locally) infeasible. Details on the implemented restoration phase are presented in Section 3.3.

To ensure global convergence of the overall method it is sufficient to ensure global convergence for each barrier parameter with a fixed value, μ_l. Therefore, the filter $\mathcal{F}_k$ is reset to its initial definition (21), whenever μ_l is decreased. It might be possible to reset the filter in ways that include information from the previous barrier problem, but in our experience the re-initialization works well in practice.

2.4. Second-Order Corrections

Many methods for nonlinear optimization use second-order corrections (see, e.g., [7, 12]) to improve the proposed step if a trial point has been rejected. A second-order correction (SOC) for some step $\tilde{d}_k^x$ aims to reduce the infeasibility by applying an additional Newton-type step for the constraints at the point $x_k + \tilde{d}_k^x$, using the Jacobian A_k^T at x_k. In the proposed method, if the first trial step size $\alpha_{k,0}$ has been rejected and if $\theta(x_k(\alpha_{k,0})) \ge \theta(x_k)$, a second-order correction $d_k^{x,\mathrm{soc}}$ (for the step $\tilde{d}_k^x = \alpha_{k,0} d_k^x$) is computed that satisfies

$$A_k^T d_k^{x,\mathrm{soc}} + c(x_k + \alpha_{k,0} d_k^x) = 0. \tag{24}$$

The new corrected search direction is then obtained from

$$d_k^{x,\mathrm{cor}} = \alpha_{k,0} d_k^x + d_k^{x,\mathrm{soc}}. \tag{25}$$

Condition (24) does not uniquely define the second-order correction, and different choices would be possible. In order to avoid additional matrix factorizations, the proposed method uses the same matrix as in (13) to compute the overall corrected step (25) from

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & A_k \\ A_k^T & -\delta_c I \end{bmatrix} \begin{pmatrix} d_k^{x,\mathrm{cor}} \\ d_k^\lambda \end{pmatrix} = -\begin{pmatrix} \nabla\varphi_{\mu_j}(x_k) + A_k\lambda_k \\ c_k^{\mathrm{soc}} \end{pmatrix}. \tag{26}$$

Here, we choose

$$c_k^{\text{soc}} = \alpha_{k,0} c(x_k) + c(x_k + \alpha_{k,0} d_k^x), \tag{27}$$

which is obtained from (13), (24) and (25).

Once the corrected search direction $d_k^{x,\text{cor}}$ has been computed, we again apply the fraction-to-the-boundary rule

$$\alpha_k^{\text{soc}} := \max \left\{ \alpha \in (0, 1] : x_k + \alpha d_k^{x,\text{cor}} \geq (1 - \tau_j) x_k \right\} \tag{28}$$

and check if the resulting trial point $x_k^{\text{soc}} := x_k + \alpha_k^{\text{soc}} d_k^{x,\text{cor}}$ is acceptable to the filter and satisfies the filter acceptance criteria. Note that the original search direction d_k^x is still used in (19) and the right hand side of (20). Also, x_k^{soc} replaces $x(\alpha_k)$ in (20).

If this trial point passes the tests, it is accepted as the new iterate. Otherwise, we apply additional second-order corrections, unless the correction step has not decreased the constraint violation by a fraction $\kappa_{\text{soc}} \in (0, 1)$ or a maximum number $p^{\max}$ of second-order corrections has been performed. In that case, the original search direction d_k^x is restored and the regular backtracking line search is resumed with a shorter step size $\alpha_{k,1} = \frac{1}{2}\alpha_{k,0}$.

Note that by choosing to apply the second-order correction at the step $\tilde{d}_k^x = \alpha_{k,0} d_k^x$ instead of, say, the full step d_k^x, no additional evaluation of the constraints is required. This also guarantees that the constraints are never evaluated for arguments violating the bound constraints (2c), at which they might not be defined.

2.5. The Algorithm

Next we formally state the overall filter line-search algorithm for solving the barrier problem (3).

Algorithm A (LINE-SEARCH FILTER BARRIER METHOD).
Given: Starting point (x_0, λ_0, z_0) with $x_0, z_0 > 0$; initial value for the barrier parameter $\mu_0 > 0$; constants $\epsilon_{\text{tol}} > 0$; $s_{\max} \geq 1$; $\kappa_\epsilon > 0$; $\kappa_\mu \in (0, 1)$; $\theta_\mu \in (1, 2)$; $\tau_{\min} \in (0, 1)$; $\kappa_\Sigma > 1$; $\theta_{\max} \in (\theta(x_0), \infty]$; $\theta_{\min} > 0$; $\gamma_\theta, \gamma_\varphi \in (0, 1)$; $\delta > 0$; $\gamma_\alpha \in (0, 1]$; $s_\theta > 1$; $s_\varphi \geq 1$; $\eta_\varphi \in (0, \frac{1}{2})$; $\kappa_{\text{soc}} \in (0, 1)$; $p^{\max} \in \{0, 1, 2, \dots\}$.

A-1. *Initialize.* Initialize the iteration counters $j \leftarrow 0$ and $k \leftarrow 0$, as well as the filter $\mathcal{F}_0$ from (21). Obtain τ_0 from (8).
A-2. *Check convergence for the overall problem.* If $E_0(x_k, \lambda_k, z_k) \leq \epsilon_{\text{tol}}$ (with the error estimate E_0 defined in (5)), then STOP [CONVERGED].
A-3. *Check convergence for the barrier problem.* If $E_{\mu_j}(x_k, \lambda_k, z_k) \leq \kappa_\epsilon \mu_j$, then:
  A-3.1. Compute μ_{j+1} and τ_{j+1} from (7) and (8), and set $j \leftarrow j + 1$;
  A-3.2. Re-initialize the filter $\mathcal{F}_k \leftarrow \{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq \theta^{\max}\}$;
  A-3.3. If $k = 0$ repeat this Step A-3, otherwise continue at A-4.
A-4. *Compute the search direction.* Compute $(d_k^x, d_k^\lambda, d_k^z)$ from (13), where δ_w and δ_c are obtained from Algorithm IC described in Section 3.1.
A-5. *Backtracking line search.*

A-5.1. *Initialize the line search.* Set $\alpha_{k,0} = \alpha_k^{\max}$ with $\alpha_k^{\max}$ from (15a), and set $l \leftarrow 0$.

A-5.2. *Compute the new trial point.* Set $x_k(\alpha_{k,l}) := x_k + \alpha_{k,l} d_k^x$.

A-5.3. *Check acceptability to the filter.* If $(\theta(x_k(\alpha_{k,l})), \varphi_{\mu_j}(x_k(\alpha_{k,l}))) \in \mathcal{F}_k$, reject the trial step and go to Step A-5.5.

A-5.4. *Check sufficient decrease with respect to the current iterate.*
- *Case I:* $\theta(x_k) \le \theta^{\min}$ *and* (19) *holds:* If (20) holds, accept the trial step $x_{k+1} := x_k(\alpha_{k,l})$ and go to A-6. Otherwise, continue at A-5.5.
- *Case II:* $\theta(x_k) > \theta^{\min}$ *or* (19) *is not satisfied:* If (18) holds, accept the trial step $x_{k+1} := x_k(\alpha_{k,l})$ and go to A-6. Otherwise, continue at A-5.5.

A-5.5. *Initialize the second-order correction.* If $l > 0$ or $\theta(x_{k,0}) < \theta(x_k)$, skip the second-order correction (SOC) and continue at A-5.10. Otherwise, initialize the SOC counter $p \leftarrow 1$ and c_k^{soc} from (27). Initialize $\theta_{\text{old}}^{\text{soc}} \leftarrow \theta(x_k)$.

A-5.6. *Compute the second-order correction.* Compute $d_k^{x,\text{cor}}$ and d_k^{λ} from (26), α_k^{soc} from (28), and $x_k^{\text{soc}} := x_k + \alpha_k^{\text{soc}} d_k^{x,\text{cor}}$.

A-5.7. *Check acceptability to the filter (in SOC).* If $(\theta(x_k^{\text{soc}}), \varphi_{\mu_j}(x_k^{\text{soc}})) \in \mathcal{F}_k$, reject the trial step size and go to Step A-5.10.

A-5.8. *Check sufficient decrease with respect to the current iterate (in SOC).*
- *Case I:* $\theta(x_k) \le \theta^{\min}$ *and* (19) *holds (for* $\alpha_{k,0}$*):* If (20) holds with "$x_k(\alpha_{k,l})$" replaced by "x_k^{soc}", accept the trial step $x_{k+1} := x_k^{\text{soc}}$ and go to A-6. Otherwise, continue at A-5.9.
- *Case II:* $\theta(x_k) > \theta^{\min}$ *or* (19) *is not satisfied (for* $\alpha_{k,0}$*):* If (18) holds with "$x_k(\alpha_{k,l})$" replaced by "x_k^{soc}", accept the trial step $x_{k+1} := x_k^{\text{soc}}$ and go to A-6. Otherwise, continue at A-5.9.

A-5.9. *Next second-order correction.* If $p = p^{\max}$ or $\theta(x_k^{\text{soc}}) > \kappa_{\text{soc}} \theta_{\text{old}}^{\text{soc}}$, abort SOC and continue at A-5.10. Otherwise, increase the SOC counter $p \leftarrow p + 1$, and set $c_k^{\text{soc}} \leftarrow \alpha_k^{\text{soc}} c_k^{\text{soc}} + c(x_k^{\text{soc}})$ and $\theta_{\text{old}}^{\text{soc}} \leftarrow \theta(x_k^{\text{soc}})$. Go back to A-5.6.

A-5.10. *Choose the new trial step size.* Set $\alpha_{k,l+1} = \frac{1}{2}\alpha_{k,l}$ and $l \leftarrow l+1$. If the trial step size becomes too small, i.e., $\alpha_{k,l} < \alpha_k^{\min}$ with $\alpha_k^{\min}$ defined in (23), go to the feasibility restoration phase in A-9. Otherwise, go back to A-5.2.

A-6. *Accept the trial point.* Set $\alpha_k := \alpha_{k,l}$ (or $\alpha_k := \alpha_k^{\text{soc}}$ if the SOC point was accepted in A-5.8), and update the multiplier estimates λ_{k+1} and z_{k+1} from (14b) and (14c) with α_k^z from (15b). Apply (16) to correct z_{k+1} if necessary.

A-7. *Augment the filter if necessary.* If (19) or (20) do not hold for α_k, augment the filter using (22). Otherwise leave the filter unchanged, i.e., set $\mathcal{F}_{k+1} := \mathcal{F}_k$.

A-8. *Continue with the next iteration.* Increase the iteration counter $k \leftarrow k + 1$ and go back to A-2.

A-9. *Feasibility restoration phase.* Augment the filter using (22), and compute a new iterate $x_{k+1} > 0$ by decreasing the infeasibility measure $\theta(x)$, so that x_{k+1} is acceptable to the augmented filter, i.e., $(\theta(x_{k+1}), \varphi_{\mu_j}(x_{k+1})) \notin \mathcal{F}_{k+1}$. Then continue with the regular iteration in Step A-8.

If the evaluation of the objective function f or constraint functions c results in an error (such as `NaN`, "Not a Number", or `Inf`, "Infinity") for a trial point $x_k(\alpha_{k,l})$, the step size is immediately rejected, and the backtracking algorithm continues in Step A-5.10.

Note that in each iteration at least one trial point will be tested before the algorithm may switch to the restoration phase. Also, the condition in Step A-3.3 ensures that eventually at least one step is taken for each decreased value of the barrier parameter. This is necessary to achieve fast local convergence in the neighborhood of a local solution satisfying the strong second-order optimality conditions [4].

In our implementation, the ℓ_1 norm is used to measure the infeasibility, i.e., $\theta(x) = \|c(x)\|_1$. The values of the constants in our implementation (if their value has not yet been mentioned earlier) are $\kappa_\epsilon = 10$; $\kappa_\mu = 0.2$; $\theta_\mu = 1.5$; $\tau_{\min} = 0.99$; $\gamma_\theta = 10^{-5}$; $\gamma_\varphi = 10^{-5}$; $\delta = 1$; $\gamma_\alpha = 0.05$; $s_\theta = 1.1$; $s_\varphi = 2.3$; $\eta_\varphi = 10^{-4}$; $\kappa_{\text{soc}} = 0.99$; $p^{\max} = 4$; as well as $\mu_0 = 0.1$, $\theta^{\max} = 10^4 \max\{1, \theta(x_0)\}$ and $\theta^{\min} = 10^{-4} \max\{1, \theta(x_0)\}$, where x_0 is the starting point. These values have been chosen because they seem to produce overall good performance, compared to other values we have explored. But the most efficient choice of the values of those numerical parameters are usually problem dependent. The numerical results in Section 4 were obtained with the tolerance $\epsilon_{\text{tol}} = 10^{-8}$ (which is approximately $\sqrt{\epsilon_{\text{mach}}}$).

3. Details of the Implementation

3.1. Inertia Correction

In order to be able to compute the search direction from (11), we need to ensure that the iteration matrix is non-singular. In addition, as mentioned earlier, the filter line-search method requires that the matrix in the top-left block of (11), projected onto the null space of the constraint Jacobian A_k^T, is positive definite[1]. These conditions are satisfied if the iteration matrix has the inertia $(n, m, 0)$, i.e., if it has exactly n positive, m negative, and no zero eigenvalues [20]. Therefore, if the inertia of this matrix is not $(n, m, 0)$, the linear system (13) is re-solved in our implementation with a modified iteration matrix for different trial values for the scalars $\delta_w, \delta_c \geq 0$ until the inertia is as desired. The inertia of the iteration matrix is readily available from several symmetric indefinite linear solvers such as the factorization routine MA27 from the Harwell subroutine library [19] used in our implementation.

Note that the desired inertia is obtained if δ_w is sufficiently large and the constraint Jacobian $\nabla c(x_k)^T$ has full rank. If $\nabla c(x_k)^T$ is rank-deficient, the matrix is singular as long as δ_c is zero, but a positive value for δ_c and a sufficiently large value of δ_w gives the correct eigenvalue signatures[2]. In practice, however, the iteration matrix can become so ill-conditioned that the factorization cannot be performed successfully, even with very large values of δ_w and some $\delta_c > 0$. In this case, we give up on the current step computation and switch directly to the feasibility restoration phase, hoping that the matrix has better properties close to feasible points.

[1] The global convergence proof in [26] requires that the eigenvalues of the projection are uniformly bounded away from zero. However, since guaranteeing this property does not seem to be possible without considerable computational effort, e.g., construction of the projected matrix explicitly, followed by an eigenvalue decomposition, we only guarantee positive definiteness in each iteration.

[2] The minus sign for the δ_c-perturbation is used to avoid generating too many positive eigenvalues.

These observations motivate the following heuristic for choosing δ_c and δ_w.

Algorithm IC (INERTIA CORRECTION).
Given: Constants $0 < \bar{\delta}_w^{\min} < \bar{\delta}_w^0 < \bar{\delta}_w^{\max}$; $\bar{\delta}_c > 0$; $0 < \kappa_w^- < 1 < \kappa_w^+ < \bar{\kappa}_w^+$; $\kappa_c \geq 0$.
Initialize $\delta_w^{\text{last}} \leftarrow 0$ at the beginning of the optimization.

In each iteration k:

IC-1. Attempt to factorize the unmodified matrix in (13) with $\delta_w = \delta_c = 0$. If the matrix is non-singular and its inertia is $(n, m, 0)$, use the resulting search direction in the line search. Otherwise continue with IC-2.
IC-2. If the iteration matrix has zero eigenvalues, set $\delta_c \leftarrow \bar{\delta}_c \mu^{\kappa_c}$, otherwise set $\delta_c \leftarrow 0$.
IC-3. If $\delta_w^{\text{last}} = 0$, set $\delta_w \leftarrow \bar{\delta}_w^0$, otherwise set $\delta_w \leftarrow \max\{\bar{\delta}_w^{\min}, \kappa_w^- \delta_w^{\text{last}}\}$.
IC-4. Attempt to factorize the modified matrix in (13). If the inertia is now $(n, m, 0)$, set $\delta_w^{\text{last}} \leftarrow \delta_w$ and use the resulting search direction in the line search. Otherwise continue with IC-5.
IC-5. If $\delta_w^{\text{last}} = 0$, set $\delta_w \leftarrow \bar{\kappa}_w^+ \delta_w$, otherwise set $\delta_w \leftarrow \kappa_w^+ \delta_w$.
IC-6. If $\delta_w > \bar{\delta}_w^{\max}$, abort the search direction computation, skip the backtracking line search, and switch directly to the restoration phase in Step A-9 of Algorithm A. Otherwise, go back to IC-4.

In our implementation, we have $\bar{\delta}_w^{\min} = 10^{-20}$, $\bar{\delta}_w^0 = 10^{-4}$, $\bar{\delta}_w^{\max} = 10^{40}$, as well as $\bar{\kappa}_w^+ = 100$, $\kappa_w^+ = 8$, $\kappa_w^- = \frac{1}{3}$ and $\kappa_c = \frac{1}{4}$. The values of $\bar{\delta}_c = 10^{-8}$ is chosen to be approximately $\sqrt{\epsilon_{\text{mach}}}$.

The above heuristic first checks in IC-1 if the unmodified matrix has the desired inertia so that the "pure" Newton search direction is used whenever possible (with an exception mentioned below). If IC-1 is unsuccessful, increasing values for δ_w are used. Note that the first trial value is based on δ_w^{last}, which stores the perturbation value from the last time a modification of the iteration matrix was necessary. In this way, we attempt to find the smallest perturbation necessary (within some factor) while at the same time avoiding futile factorizations in IC-4 for values of δ_w that are too small. Here we assume that the minimum necessary perturbation is of the same order of magnitude in successive iterations. The reason for using a much larger factor $\bar{\kappa}_w^+$ in IC-5 for the very first necessary correction than for the correction in later iterations is that we want to avoid a high number of trial factorizations when the scale of the problem and the order of magnitude for a successful correction is not yet known. By choosing κ_w^- and κ_w^+ so that $\kappa_w^- \kappa_w^+ \neq 1$ we avoid situations where the same perturbation δ_w is used in successive iterations. Otherwise, the algorithm could repeatedly produce very large steps d_k^x due to a nearly singular iteration matrix, so that only very small step sizes α_k would be taken and little progress would be made.

A nonzero value for δ_c is always chosen if the unmodified iteration matrix has a zero eigenvalue, as we assume that the singularity is caused by a rank-deficient constraint Jacobian. We do not attempt to verify whether the singularity is instead caused by a singular projected Hessian matrix, because this would increase the number of trial factorizations. Note that the nonzero value for δ_c in Step IC-2 converges to zero as $\mu \to 0$ (if $\kappa_c > 0$), so that the perturbation is smaller when a solution of the problem is approached.

In some problem instances, the iteration matrix is structurally singular, for example, when the equality constraint gradients are always linearly dependent, or when the reduced Hessian is always rank deficient. We therefore deviate from Algorithm IC in our implementation, if the iteration matrix is singular in the first three iterations and if this can be corrected by choosing a positive value for δ_c. In that case, the value of δ_c used in the later iteration will always be $\bar{\delta}_c \mu^{\kappa_c}$ (also in IC-1) in order to avoid futile factorizations with $\delta_c = 0$. Similarly, if in the first three iterations singularity of the iteration matrix can be avoided by choosing $\delta_w > 0$, we assume that a correction $\delta_w > 0$ is necessary in any case, and Step IC-1 above is executed with $\delta_w = \max\{\bar{\delta}_w^{\min}, \kappa_w^- \delta_w^{\text{last}}\}$.

We note that the optimization code LOQO [23] also uses a similar trial procedure to find an appropriate perturbation of the Hessian.

3.2. Two Accelerating Heuristics

One possible pitfall of the filter method described in Section 2.3 is that the filter $\mathcal{F}_k$ in the current iteration might include (θ, φ)-pairs that have been added earlier for an iterate in a different region, with similar values for $\varphi_{\mu_j}(x)$ and $\theta(x)$ (see also [26, Remark 7]). This could prevent the algorithm from taking good steps toward a nearby local solution. As a result, the backtracking line-search algorithm might repeatedly cut back the step size, or could be forced unnecessarily to resort to the feasibility restoration phase.

We also noticed that in some cases the full step (with $\alpha_{k,0} = \alpha_k^{\max}$, even with a second-order correction) is rejected in successive iterations, because it does not achieve sufficient progress with respect to the current iterate (condition (18) or (20)). This causes the algorithm to make little progress, even though the method may converge faster when the acceptance of the full step is temporarily allowed.

In order to avoid these inefficiencies, two heuristics are added to the proposed method. The algorithm counts the number of successive iterations in which the first trial step (including a second-order correction) is rejected. If this number exceeds a given threshold (four in our implementation), then one of the following actions are taken after the last of those iterations, say iteration k:

- *Case I: If $\theta^{\max} > \theta(x_{k+1})/10$, and the last unsuccessful trial step size in the backtracking line search was rejected in A-5.3 because the trial point was not acceptable to the filter.*
  In this case, the current filter might be blocking good progress, caused by historic information from iterates in a different (and now irrelevant) region of $\mathbb{R}^n$. To avoid further inefficiencies, the filter is re-initialized for the next iteration by setting $\mathcal{F}_{k+1} = \{(\theta, \varphi) : \theta \geq \theta^{\max}\}$ in Step A-7, after the maximal permitted constraint violation has been been reduced, i.e., $\theta^{\max} \leftarrow 0.1\theta^{\max}$. Note that the decrease of $\theta^{\max}$ ensures that the filter is not reset infinitely many times, unless the infeasibility becomes arbitrarily small.
- *Case II: Otherwise.*
  Here, we hope to overcome possible inefficiencies by tentatively ignoring the filter criteria for one iteration, similar to a watchdog procedure [5] (with one relaxed step). In the next iteration, $k + 1$, we choose $\alpha_{k+1} = \alpha_{k+1}^{\max}$ without any backtracking line search. The filter is not augmented in Step A-7 for iteration $k + 1$, and the search

directions $d^x_{k+1}, d^\lambda_{k+1}, d^z_{k+1}$ are stored as a backup. Then, new search directions d^x_{k+2} etc. are computed at the point $x_{k+2} = x_{k+1} + \alpha^{\max}_{k+1} d^x_{k+1}$ etc. We check whether the trial point for the full step $\alpha^{\max}_{k+2}$ is acceptable to the filter $\mathcal{F}_{k+1}$ and satisfies the line-search acceptance criteria *for the previous iteration* $k+1$, i.e., whether "$\theta(x_{k+2} + \alpha^{\max}_{k+2} d^x_{k+2}) \le (1-\gamma_\theta)\theta(x_{k+1})$" (similar to (18b)) or

$$\varphi_{\mu_j}(x_{k+2} + \alpha^{\max}_{k+2} d^x_{k+2}) \le \varphi_{\mu_j}(x_{k+1}) + \eta_\varphi \alpha^{\max}_{k+1} \nabla\varphi_{\mu_j}(x_{k+1})^T d^x_{k+1},$$

(depending on the switching condition in Step A-5.4 for iteration $k+1$). If these tests are passed, the trial point is accepted as iterate x_{k+3}, and λ_{k+3} and z_{k+3} are updated accordingly. In this case, we have made sufficient progress with respect to x_{k+1} within two iterations, and the filter is augmented using (22) with x_{k+1}, if (19) or (20) does not hold (for x_{k+1} and d^x_{k+1}). If the tests fail, the tentative iterate x_{k+2} is abandoned, the original search directions d^x_{k+1}, d^λ_{k+1}, d^z_{k+1} are restored, and the usual backtracking line-search procedure *from* x_{k+1} is resumed to produce a new iterate $x_{k+3} = x_{k+1} + \alpha_{k+1,l} d^x_{k+1}$.

Even though these heuristics are not frequently activated and the watchdog heuristic might in some cases increase the number of iterations, they appear to have an overall positive effect.

3.3. Feasibility Restoration Phase

A key ingredient of the filter line-search method is the feasibility restoration phase (see Step A-9). The task of the restoration phase is to compute a new iterate acceptable to the augmented filter $\mathcal{F}_{k+1}$ by decreasing the infeasibility, whenever the regular backtracking line-search procedure cannot make sufficient progress and the step size becomes too small (see Step A-5.10). In addition, as mentioned in Section 3.1, the method switches to the restoration phase whenever the linear system (13) is very ill-conditioned and cannot be factorized successfully despite modifications of the iteration matrix. In summary, the feasibility restoration phase is very important in the sense that it is invoked whenever the progress to the solution becomes difficult, and hence it needs to be very robust.

The feasibility restoration phase has another significant purpose, namely to detect (local) infeasibility. Infeasible problems arise, for example, due to modeling errors, and a user should be notified quickly of a badly-posed problem. If the problem is infeasible, the algorithm is ultimately not able to generate sufficient progress in the regular backtracking line-search procedure and reverts to the restoration phase. We would then want the restoration phase to converge to a non-zero minimizer of the constraint violation (in some norm), and in this way to provide an indication of infeasibility.

We note that for the global convergence proof of the filter line-search method in [26] it is assumed that, in the neighborhood of feasible points, the gradients of the active constraints are linearly independent. It is shown in [26] that as a consequence the algorithm does not switch to the feasibility restoration phase at (almost) feasible points. However, in practice this assumption might be violated, and the restoration phase might be called at a point with a very small (or zero) value of θ. Since further reduction of the infeasibility might then be difficult and not lead to progress in the optimization process, the current

implementation of the algorithm terminates with an error message, if the restoration phase is called at a point x_k with $\theta(x_k) < \epsilon_{\text{tol}}$.

Our "regular" restoration phase algorithm is described next. An alternative method is discussed in Section 3.3.2. These algorithms are also iterative methods. In order to avoid confusion, we use overbars (such as $\bar{x}$) to denote quantities referring to the restoration phase and use the subscript t for the restoration phase iteration counter.

3.3.1. Minimization of the Constraint Violation In this section we describe the first algorithm for the restoration phase. The goal of this method is to return a new iterate $x_{k+1} > 0$ with $(\theta(x_{k+1}), \varphi_{\mu_j}(x_{k+1})) \notin \mathcal{F}_{k+1}$ for Step A-9, or to converge to a non-zero minimizer (or at least a stationary point) of some norm of the constraint violation. The restoration phase algorithm applies the primal-dual interior-point filter line-search algorithm outlined in the previous sections to a smooth reformulation of the optimization problem

$$\min_{\bar{x}\in\mathbb{R}^n} \quad \|c(\bar{x})\|_1 + \frac{\zeta}{2}\|D_R(\bar{x} - \bar{x}_R)\|_2^2 \tag{29a}$$

$$\text{s.t.} \quad \bar{x} \geq 0. \tag{29b}$$

Here, a term is included in the objective function that penalizes the deviation from a reference point $\bar{x}_R$, where $\zeta > 0$ is the weighting parameter, and the scaling matrix D_R is defined by

$$D_R = \text{diag}(\min\{1, 1/|\bar{x}_R^{(1)}|\}, \ldots, \min\{1, 1/|\bar{x}_R^{(n)}|\}).$$

The reference point $\bar{x}_R$ is chosen to be the iterate x_k at which the restoration phase is called in Step A-9. In this way, we seek to decrease the constraint violation but try to avoid a large deviation from $\bar{x}_R$ and an undesired significant increase in the barrier objective function φ_{μ_j}. A related restoration phase problem formulation that attempts to minimize the constraint violation and also includes a regularization term based on $\bar{x} - \bar{x}_R$ has been proposed by Ulbrich et al. [22].

A smooth reformulation of (29) is obtained by introducing non-negative variables $\bar{p}, \bar{n} \in \mathbb{R}^m$ that capture the positive and negative parts of the constraints,

$$\min_{\bar{x}\in\mathbb{R}^n, \bar{p}, \bar{n}\in\mathbb{R}^m} \quad \sum_{i=1}^{m} \left(\bar{p}^{(i)} + \bar{n}^{(i)}\right) + \frac{\zeta}{2}\|D_R(\bar{x} - \bar{x}_R)\|_2^2 \tag{30a}$$

$$\text{s.t.} \quad c(\bar{x}) - \bar{p} + \bar{n} = 0 \tag{30b}$$

$$\bar{x}, \bar{p}, \bar{n} \geq 0. \tag{30c}$$

This nonlinear optimization problem is of the form (2). We can therefore apply the "regular" interior-point algorithm described in the earlier sections and solve a sequence of barrier problems

$$\min_{\bar{x}\in\mathbb{R}^n,\bar{p},\bar{n}\in\mathbb{R}^m} \quad \rho\sum_{i=1}^{m}\left(\bar{p}^{(i)}+\bar{n}^{(i)}\right)+\frac{\zeta}{2}\|D_R(\bar{x}-\bar{x}_R)\|_2^2-\bar{\mu}\sum_{i=1}^{n}\ln(\bar{x}^{(i)})$$
$$-\bar{\mu}\sum_{i=1}^{m}\ln(\bar{p}^{(i)})-\bar{\mu}\sum_{i=1}^{m}\ln(\bar{n}^{(i)}) \tag{31a}$$
$$\text{s.t.}\quad c(\bar{x})-\bar{p}+\bar{n}=0 \tag{31b}$$

with the filter line-search procedure. We introduced the additional scaling parameter $\rho > 0$ in order to allow a relative scaling of the overall objective function (31a) with respect to the constraints (31b). By default, the parameter ρ is chosen to be 1000, which seems to work well in practice.

Note that if the regularization parameter $\zeta > 0$ is chosen sufficiently small, the optimization problem (30) is the exact penalty formulation [12] of the problem "find the feasible point that is closest (in a weighted norm) to the reference point $\bar{x}_R$,"

$$\min_{\bar{x}\in\mathbb{R}^n} \quad \|D_R(\bar{x}-\bar{x}_R)\|_2^2$$
$$\text{s.t.}\quad c(\bar{x})=0, \quad \bar{x}\geq 0.$$

This appears to be an intuitive goal for the restoration phase. An additional desired consequence of the penalty term is that the optimal solution of (30) is usually a strict local solution, which makes this nonlinear optimization problem easier to solve. This would usually not be the case for the choice $\zeta = 0$, because then all points in the manifold defined by "$c(x) = 0$" would be minimizers. Since a sufficiently small value of the regularization parameter ζ is not known before a solution of (29) is determined, we choose $\zeta = \sqrt{\bar{\mu}}$, so that ζ is driven to zero together with $\bar{\mu}$.

In addition to the original variables $\bar{x}$, the barrier problem (31) contains the variables $\bar{p}$ and $\bar{n}$, and the corresponding primal-dual equations (similar to (4)) include their accompanying dual variables, say $\bar{z}_p$ and $\bar{z}_n$. Search directions for the line search are, as before, obtained by linearization of these equations. Some straight-forward algebraic manipulations show that they can be computed from (omitting the iteration index t for simplicity)

$$\begin{bmatrix} \bar{W}+\zeta D_R^2+\bar{\Sigma} & \nabla c(\bar{x}) \\ \nabla c(\bar{x})^T & -\bar{\Sigma}_p^{-1}-\bar{\Sigma}_n^{-1}\end{bmatrix}\begin{pmatrix}\bar{d}^x\\ \bar{d}^\lambda\end{pmatrix}=$$
$$-\begin{pmatrix}\zeta D_R^2(\bar{x}-\bar{x}_R)+\nabla c(\bar{x})\bar{\lambda}-\bar{\mu}\bar{X}^{-1}e\\ c(\bar{x})-\bar{p}+\bar{n}+\rho\bar{Z}_p^{-1}(\bar{\mu}e-\bar{p})+\rho\bar{Z}_n^{-1}(\bar{\mu}e-\bar{n})\end{pmatrix}, \tag{32}$$

where $\bar{W}=\sum_{i=1}^{m}\bar{\lambda}^{(i)}\nabla^2_{xx}c(\bar{x})$, $\bar{\Sigma}=\bar{X}^{-1}\bar{Z}$, $\bar{\Sigma}_p=\bar{P}^{-1}\bar{Z}_p$, and $\bar{\Sigma}_n=\bar{N}^{-1}\bar{Z}_n$. Subsequently, $\bar{d}^p$, $\bar{d}^n$, $\bar{d}^{z_p}$ and $\bar{d}^{z_n}$ are obtained from

$$\bar{d}^p=\bar{Z}_p^{-1}(\bar{\mu}e+\bar{P}(\bar{\lambda}+\bar{d}^\lambda)-\rho\,\bar{p}), \qquad \bar{d}^{z_p}=\bar{\mu}\bar{P}^{-1}e-\bar{z}^p-\bar{\Sigma}_p\bar{d}^p,$$
$$\bar{d}^n=\bar{Z}_n^{-1}(\bar{\mu}e-\bar{N}(\bar{\lambda}+\bar{d}^\lambda)-\rho\,\bar{n}), \qquad \bar{d}^{z_n}=\bar{\mu}\bar{N}^{-1}e-\bar{z}^n-\bar{\Sigma}_n\bar{d}^n,$$

and $\bar{d}^z$ from

$$\bar{d}^z=\bar{\mu}\bar{X}^{-1}e-\bar{z}-\bar{\Sigma}\bar{d}^x.$$

Note that the structure of the nonzero elements of the linear system in (32) is identical to the one in (13), which allows us to use the same code (and symbolic factorization) in the step computations as for the regular iteration, including the Hessian correction mechanism described in Section 3.1. Here, we keep $\delta_c = 0$ at all times since the Jacobian of the constraint (31b) cannot be rank-deficient. We note that second-order corrections as described in Section 2.4 have not been implemented for the restoration phase.

The filter line-search method applied to (31) might itself revert to a restoration phase. If this occurs, we compute the optimal solution of (31) *for a fixed value of* $\bar{x}$ (namely the current iterate $\bar{x}_t$) and use this as the "result" of the restoration phase within the restoration phase. Since (31) then becomes separable, this can easily be done by solving a quadratic equation for each $(\bar{p}^{(i)}, \bar{n}^{(i)})$ pair, that is

$$\bar{n}^{(i)} = \frac{\bar{\mu} - \rho\, c^{(i)}(\bar{x})}{2\rho} + \sqrt{\left(\frac{\bar{\mu} - \rho\, c^{(i)}(\bar{x})}{2\rho}\right)^2 + \frac{\bar{\mu}\, c^{(i)}(\bar{x})}{2\rho}} \qquad \bar{p}^{(i)} = c^{(i)}(\bar{x}) + \bar{n}^{(i)}. \tag{33}$$

Since the gradients of the constraints (31b) always have full rank, the analysis in [26] shows that the restoration phase (within the restoration phase filter algorithm) is not invoked at a feasible point (for (31)).

At the beginning of the restoration phase algorithm, the first barrier parameter $\bar{\mu}_0$ is chosen to be the maximum of the current barrier parameter, μ_j, of the regular iteration and $\|c(x_k)\|_\infty$. The initial value for $\bar{x}_0$ is simply chosen as the regular iterate x_k at which the restoration phase is called (identical to $\bar{x}_R$). To initialize the dual variables we set $\bar{\lambda}_0 = 0$ and $\bar{z}_0^{(i)} = \min\{\rho, z_k^{(i)}\}$, for $i = 1, \ldots, n$. Furthermore, $\bar{p}_0$ and $\bar{n}_0$ are computed from (33), and their dual variables are initialized as $\bar{z}_{p,0} = \bar{\mu}(\bar{P}_0)^{-1}e$ and $\bar{z}_{n,0} = \bar{\mu}(\bar{N}_0)^{-1}e$. In this way, the optimality conditions for the variables added for the restoration phase problem are all satisfied at the starting point, so that the first restoration phase step usually tends to reduce $\theta(x) = \|c(x)\|_1$ without being "distracted" by the introduction of the new variables.

The restoration phase is discontinued as soon as (i) the current restoration phase iterate, say $\bar{x}_t$, is acceptable for the augmented regular filter (i.e., $(\theta(\bar{x}_t), \varphi_{\mu_j}(\bar{x}_t)) \notin \mathcal{F}_{k+1}$, see Step A-9) and (ii) $\theta(x_t) \leq \kappa_{\text{resto}}\theta(\bar{x}_R)$ for some constant $\kappa_{\text{resto}} \in (0, 1)$ ($\kappa_{\text{resto}} = 0.9$ in our implementation). The motivation for the second condition is to ensure that once the restoration phase is activated, reasonable progress toward feasibility is achieved; this has proven advantageous in our numerical experiments. The regular method is then resumed from $x_{k+1} = \bar{x}_t$. Note, that because an interior-point algorithm is used to solve (30), it is guaranteed that $\bar{x}_t > 0$. In order to compute a step for the bound multipliers z after the return to the regular method, we pretend that the entire progress during the restoration phase was one single step, $d_k^x := x_{k+1} - x_k$, and obtain z_{k+1} from (12), (14c) and (15b). The equality constraint multipliers are re-initialized as described in Section 3.6 below.

On the other hand, if the termination criterion for the restoration phase problem, similar to (6), is satisfied before the regular method can be resumed, the proposed algorithm terminates with the message that the problem seems locally infeasible.

3.3.2. Reducing the KKT Error As mentioned earlier in Section 3.2, "historic" information in the filter $\mathcal{F}_k$ originating from points in a different region of $\mathbb{R}^n$ can prevent fast progress in the neighborhood of a local solution (x_*, λ_*, z_*). The heuristics in Section 3.2 might not always be able to overcome this difficulty, so that eventually the restoration phase might be invoked. However, the regular iteration steps are Newton(-type) steps for the primal-dual equations, and should therefore be taken close to (x_*, λ_*, z_*).

Therefore, we do not immediately revert to the algorithm described in Section 3.3.1 when the restoration phase is called in Step A-9. Instead, we try to achieve reduction in the norm of the primal-dual equations, using the regular iteration steps (as proposed in [26, Remark 8]). In the following description of this procedure, $F_\mu(x, \lambda, z)$ denotes the nonlinear system of equations on the left hand side of (4).

Algorithm KKT ERROR REDUCTION.
Given: Constant $\kappa_F \in (0, 1)$ ($\kappa_F = 0.999$ in our implementation).

R-0. Initialize the restoration phase iteration counter $t \leftarrow 0$ and choose the current "regular" iterate as starting point: $(\bar{x}_0, \bar{\lambda}_0, \bar{z}_0) = (x_k, \lambda_k, z_k)$.

R-1. Compute a search direction $(\bar{d}_t^x, \bar{d}_t^\lambda, \bar{d}_t^z)$ using the regular iteration matrix from (11)–(12) (with the appropriate substitutions). The modifications described in Section 3.1 are applied. Note that for $t = 0$ this search direction has already been computed in the regular iteration.

R-2. Apply the fraction-to-the-boundary rule

$$\bar{\beta}_t := \max\left\{\beta \in (0, 1] : \bar{x}_t + \beta \bar{d}_t^x \geq (1 - \tau_j)\bar{x}_t \text{ and } \bar{z}_t + \beta \bar{d}_t^z \geq (1 - \tau_j)\bar{z}_t\right\}.$$

R-3. Test whether

$$\left\| F_\mu\left(\bar{x}_{t+1}, \bar{\lambda}_{t+1}, \bar{z}_{t+1}\right)\right\|_1 \leq \kappa_F \left\| F_\mu\left(\bar{x}_t, \bar{\lambda}_t, \bar{z}_t\right)\right\|_1$$

with

$$\left(\bar{x}_{t+1}, \bar{\lambda}_{t+1}, \bar{z}_{t+1}\right) = \left(\bar{x}_t, \bar{\lambda}_t, \bar{z}_t\right) + \beta_t \left(\bar{d}_t^x, \bar{d}_t^\lambda, d_t^z\right).$$

If the evaluation of the functions at the trial point results in an error, or if this decrease condition is not satisfied, discard the trial point and switch to the robust restoration phase algorithm described in Section 3.3.1, using $\bar{x}_t$ as the reference point in the initialization of the robust restoration phase.

R-4. If $(\theta(\bar{x}_{t+1}), \varphi_{\mu_j}(\bar{x}_{t+1})) \notin \mathcal{F}_{k+1}$, continue the regular interior-point method from the point $(x_{k+1}, \lambda_{k+1}, z_{k+1}) := (\bar{x}_{t+1}, \bar{\lambda}_{t+1}, \bar{z}_{t+1})$. Otherwise, set $t \leftarrow t + 1$ and continue with Step R-1.

In the neighborhood of a strict local solution satisfying the second-order sufficient optimality conditions for the barrier problem, the projection of the Hessian $W_t + \Sigma_t$ onto the null space of the constraint Jacobian $\nabla c(\bar{x}_t)^T$ is positive definite, and therefore no modification of the iteration matrix, as described in Section 3.1, is applied. As a consequence, the search directions computed from (11)–(12) are the Newton steps for (4), so that the above procedure will accept those steps and quickly converge toward this solution, if it is started sufficiently close.

Since the norm of the KKT conditions is decreased by at least a fixed fraction, κ_F, it is guaranteed that the method eventually either resumes the regular procedure, Algorithm A, or reverts to the restoration phase described in Section 3.3.1.

The above algorithm is not attempted, if the restoration phase is triggered in the regular method because of numerical problems during the solution of the linear system (11) in Step IC-6. In that case, the method immediately proceeds to the restoration phase described in Section 3.3.1.

3.4. General Lower and Upper Bounds

For simplicity, the algorithm has been described for solving optimization problems of the form (2), but it is straight-forward to generalize the procedures outlined so far to the more general formulation (1). In particular, the resulting barrier problem then becomes

$$\min_{x\in\mathbb{R}^n} \ \varphi_{\mu_j}(x) = f(x) - \mu_j \sum_{i\in I_L} \ln\left(x^{(i)} - x_L^{(i)}\right) - \mu_j \sum_{i\in I_U} \ln\left(x_U^{(i)} - x^{(i)}\right) \quad (34a)$$

$$\text{s.t.} \ \ c(x) = 0 \quad (34b)$$

where $I_L = \{i : x_L^{(i)} \neq -\infty\}$ and $I_U = \{i : x_U^{(i)} \neq \infty\}$. Bound multipliers $z_L^{(i)}$ and $z_U^{(i)}$ are introduced for all finite lower and upper bounds, and the primal-dual Hessian Σ_k of the barrier terms is defined as the sum of $\Sigma_k^L = \text{diag}(\sigma_{k,1}^L, \ldots, \sigma_{k,n}^L)$ and $\Sigma_k^U = \text{diag}(\sigma_{k,1}^U, \ldots, \sigma_{k,n}^U)$, where

$$\sigma_{k,i}^L = \begin{cases} z_{L,k}^{(i)} / \left(x_k^{(i)} - x_L^{(i)}\right) & \text{if } i \in I_L \\ 0 & \text{otherwise} \end{cases},$$

$$\sigma_{k,i}^U = \begin{cases} z_{U,k}^{(i)} / \left(x_U^{(i)} - x_k^{(i)}\right) & \text{if } i \in I_U \\ 0 & \text{otherwise} \end{cases}.$$

For completeness, we define $z_{L,k}^{(i)} = 0$ for $i \notin I_L$ and $z_{U,k}^{(i)} = 0$ for $i \notin I_U$.

If the given lower and upper bounds for a variable are identical, this component of x is fixed to this value for all function evaluations and removed from the problem statement.

3.5. Handling Problems Without a Strict Relative Interior

As a barrier method, the proposed algorithm relies on the existence of a strict relative interior of the feasible region, i.e., of points x with $x_L < x < x_U$ and $c(x) = 0$, since otherwise a solution to the barrier problem (34) does not exist. However, this assumption can easily be violated in practice, for example, if the equality constraints implicitly imply $x^{(i)} = x_L^{(i)}$ for some i-th component. In such a case, in the process of trying to find a feasible point for a fixed value of μ_j, the algorithm might generate a sequence of

iterates where $x_k^{(i)} - x_L^{(i)}$ becomes very small. This in turn can lead to numerical difficulties during the solution of the linear system (11), because the corresponding entry in Σ_k, which is roughly of the order of $\mu_j/(x_k^{(i)} - x_L^{(i)})^2$ (see (17)), becomes very large.

As a remedy, we found it helpful to slightly relax the bounds before solving the problem by

$$x_L^{(i)} \leftarrow x_L^{(i)} - \epsilon_{\text{tol}} \max\{1, |x_L^{(i)}|\} \tag{35}$$

(similarly for x_U), in order to avoid an empty relative interior from the very beginning. Since this perturbation is of the order of the termination tolerance ϵ_{tol}, we believe that this does not constitute an unwanted modification of the problem statement.

Furthermore, the lower bound on $x^{(i)}$ is slightly relaxed by $(\epsilon_{\text{mach}})^{\frac{3}{4}} \max\{1, x_L^{(i)}\}$, whenever $x_k^{(i)} - x_L^{(i)} < \epsilon_{\text{mach}}\ \mu_j$, where ϵ_{mach} is the machine precision. An analogous procedure is applied for very small slack to upper bounds. Even if these corrections are applied repeatedly, the changes are so small that the problem statement is essentially not modified, but the numerical difficulties are usually avoided.

3.6. Initialization

Since the algorithm requires the iterates to strictly satisfy the bound constraints (1c), it is often necessary to modify the user-provided initial point so that it is sufficiently away from the boundary. For this purpose, each component i of the initial point, which has only one (say, a lower) bound, is modified by

$$x_0^{(i)} \leftarrow \max\left\{x_0^{(i)}, x_L^{(i)} + \kappa_1 \max\{1, |x_L^{(i)}|\}\right\}$$

for a constant $\kappa_1 > 0$ (similarly for variables only bounded above). The initial value of a variable $x^{(i)}$ bounded on two sides is projected into the interval $[x_L^{(i)} + p_L^{(i)}, x_U^{(i)} - p_U^{(i)}]$ with the perturbations

$$\begin{aligned} p_L^{(i)} &:= \min\{\kappa_1 \max\{1, |x_L^{(i)}|\}, \kappa_2(x_U^{(i)} - x_L^{(i)})\} \\ p_U^{(i)} &:= \min\{\kappa_1 \max\{1, |x_U^{(i)}|\}, \kappa_2(x_U^{(i)} - x_L^{(i)})\}, \end{aligned}$$

for some $\kappa_2 \in (0, \frac{1}{2})$. The default choices in our implementation are $\kappa_1 = \kappa_2 = 10^{-2}$.

The dual variables corresponding to the bound constraints are initialized to one component-wise. Finally, using the possibly modified initial point x_0 and the initial bound multipliers, the multipliers λ_0 for the equality constraints are obtained as least-square solutions for the dual infeasibility (4a), i.e., by solving the linear system

$$\begin{bmatrix} I & \nabla c(x_0) \\ \nabla c(x_0)^T & 0 \end{bmatrix} \begin{pmatrix} w \\ \lambda_0 \end{pmatrix} = -\begin{pmatrix} \nabla f(x_0) - z_{L,0} + z_{U,0} \\ 0 \end{pmatrix}, \tag{36}$$

where w is discarded after this computation. However, if λ_0 obtained in this way is too large, i.e., if $\|\lambda_0\|_\infty > \lambda_{\max}$ (with $\lambda_{\max} = 10^3$ in our implementation), the least square estimate is discarded and we set $\lambda_0 = 0$. In practice this seems to avoid poor initial guesses for λ_0 in cases where the constraint Jacobian is nearly linearly dependent at the initial point. This procedure for estimating the equality constraint multipliers is also used after the restoration phase algorithm described in Section 3.3.1 reverts to the regular method.

3.7. Handling Unbounded Solution Sets

In some cases, the set $\mathcal{S}_*$ of optimal points for (1) does not consist of isolated points, but contains an unbounded connected component. Then, the objective function of the corresponding barrier problem (34) for a fixed value of μ_j is unbounded below over the feasible set, since a log-barrier term converges to $-\infty$ as its argument goes to infinity. As a consequence, the method for solving the barrier problem might fail to converge, even though the original problem is well-posed.

In order to prevent this behavior, linear damping terms for all variables with exactly one finite bound are added to the barrier objective function (34a), which then becomes

$$\begin{aligned}\varphi_{\mu_j}(x) = f(x) - \mu_j \sum_{i\in I_L} \ln\left(x^{(i)} - x_L^{(i)}\right) - \mu_j \sum_{i\in I_U} \ln\left(x_U^{(i)} - x^{(i)}\right)\\ +\kappa_d\mu_j \sum_{i\in I_L\setminus I_U} \left(x^{(i)} - x_L^{(i)}\right) + \kappa_d\mu_j \sum_{i\in I_U\setminus I_L} \left(x_U^{(i)} - x^{(i)}\right)\end{aligned}$$

for a positive constant $\kappa_d > 0$ independent of μ_j ($\kappa_d = 10^{-4}$ in our implementation). In this way, divergence of variables that have only one bound is penalized. On the other hand, the effect of the damping term is reduced as μ_j decreases. Adding these terms to the barrier objective function corresponds to a perturbation of the dual infeasibility (4a) by $\kappa_d\mu_j e$, and the local convergence analysis [4] based on homotopy arguments still holds. In our numerical tests, this modification led to improved robustness.

3.8. Automatic Scaling of the Problem Statement

The Newton steps for the primal dual equations (4) computed from (11) are invariant to scaling of the variables, the objective and constraint functions, i.e., to replacing x, f, and c by $\tilde{x} = D_x x$, $\tilde{f}(x) = d_f f(x)$ and $\tilde{c}(x) = D_c c(x)$ for some $d_f > 0$ and positive definite diagonal matrices $D_x = \mathrm{diag}(d_x^{(1)}, \ldots, d_x^{(n)})$, $D_c = \mathrm{diag}(d_c^{(1)}, \ldots, d_c^{(m)})$. However, the overall optimization algorithm with its initialization procedures, globalization strategy and stopping criteria usually behaves very differently for different scaling factors, particularly if the scaling factors are very large or very small. In addition, numerical difficulties due to finite precision are more likely to arise if the occurring numbers are of very different orders of magnitude.

Automatic scaling of optimization problems has been examined in the past, but it is not clear how, in the nonlinear case, the variables and functions should be scaled in order to obtain good efficiency and robustness (where the sensitivities of functions with respect to changes in variables might vary drastically from one iteration to another).

In the context of this paper we take the perspective that ideally we would like to scale the variables and functions so that changing a variable by a given amount has a comparable effect on any function which depends on this variables, or in other words, so that the non-zero elements of the function gradients are of the same order of magnitude (say, 1).

We experimented with applying an equilibration algorithm (implemented in the Harwell [19] subroutines MC19 and MC29) to the first derivative matrix

$$J_0 = \begin{bmatrix} \nabla_x c(x_0)^T \\ \nabla_x f(x_0)^T \end{bmatrix}$$

to obtain scaling matrices D_x and $D_{cf} = \mathrm{diag}(D_c, d_f)$ so that the nonzero elements in $D_{cf} J_0 D_x^{-1}$ are of order one (as proposed in [6]). Similarly, we computed scaling factors so that the matrix

$$\begin{bmatrix} D_x^{-1} & 0 \\ 0 & D_c \end{bmatrix} \begin{bmatrix} \nabla^2_{xx} f(x_0) & \nabla_x c(x_0) \\ \nabla_x c(x_0)^T & 0 \end{bmatrix} \begin{bmatrix} D_x^{-1} & 0 \\ 0 & D_c \end{bmatrix}$$

has non-zero entries close to one. While these strategies seem to work well in some instances, the overall performance on the considered test set became worse. Nevertheless, these procedures are available to users of our implementation as options.

The automatic scaling procedure finally used by default in the proposed method is rather conservative and assumes that usually the given problem is well scaled and does not require modification, unless some sensitivities are large. Given a threshold value $g_{\max} > 0$ ($g_{\max} = 100$ in our implementation), we choose the scaling factors according to

$$\begin{aligned} d_f &= \min\{1, g_{\max}/\|\nabla_x f(x_0)\|_\infty\}, \\ d_c^{(j)} &= \min\{1, g_{\max}/\|\nabla_x c^{(j)}(x_0)\|_\infty\}, \qquad j = 1, \ldots, m \end{aligned}$$

and we set $D_x = I$. Note that this will never multiply a function by a number larger than one, and that all gradient components in the scaled problem are at most of the size $g_{\max}$ at the starting point.

The scaling factors are computed only at the beginning of the optimization using the starting point after the modifications described in Section 3.6.

3.9. Handling Very Small Search Directions

In a few instances we observed that the search directions d_k^x generated from (13) become very small compared to the size of the iterate x_k itself. For example, this can occur if the primal variables are already very close to their final optimal value, but the dual variables have not yet converged. We also observed this situation for very ill-scaled problems. Performing the regular backtracking line-search procedure can then be unsuccessful due to rounding errors, and can result in an unnecessary switch to the restoration phase. In order to prevent this, we allow the algorithm to take the full step with $\alpha_k = \alpha_k^{\max}$ whenever $\max\{|(d_k^x)^{(i)}|/(1 + |x_k^{(i)}|) : i = 1, \ldots, n\} < 10\epsilon_{\mathrm{mach}}$. If this is true for two consecutive iterations, the algorithm assumes that the current barrier problem has been solved as well as possible given the finite precision, and reduces the barrier parameter in A-3. If μ_j is already very small ($\epsilon_{\mathrm{tol}}/10$), the algorithm terminates with a warning message.

3.10. Numerical Issues

In our implementation of the proposed algorithm, the linear systems (13) and (32) are solved by the Harwell routine [19] MA27, after they have been equilibrated with the scaling routine MC19. As default pivot tolerance for MA27 we specify $\epsilon_{\text{piv}} = 10^{-8}$, which is about the square root of ϵ_{mach}. In our experience, it is very important to use iterative refinement in order to improve robustness of the implementation and to be able to obtain highly accurate solutions. Whereas iterative refinement on the linear systems of the form (13) itself provides somewhat better search directions than using no iterative refinement, we found that a considerable gain in robustness and precision can be achieved by applying iterative refinement on the *unreduced* non-symmetric Newton system (such as (9), but including the perturbations δ_w and δ_c). Here, we still use (13) and (12) to solve the linear system, but compute the iterative refinement residual for the larger linear system (9). This appears to be particularly important for a numerically robust implementation of the restoration phase, where iterative refinement only on (32) seems insufficient to solve the restoration phase problem to the default tolerance, even for very large pivot tolerances ϵ_{piv}. We believe that this is partly due to the fact that the diagonal elements in the smaller formulation (32) are obtained by adding numbers that may be very different in magnitude, which may lead to severe rounding error. For example, if a variable with two bounds converges to one of its bounds, then the corresponding entry in Σ_k is obtained by adding two numbers, one of which converges to zero, and the other one goes to infinity in the limit.

In addition, if the linear systems cannot be solved sufficiently well despite iterative refinement, the algorithm increases the pivot tolerance for the linear solver by $\epsilon_{\text{piv}} \leftarrow \max\{10^{-2}, \epsilon_{\text{piv}}^{3/4}\}$. Here, the pivot tolerance is increased at most once per iteration. If an increase in the pivot tolerance still does not lead to a sufficiently small residual, the search direction is used as is.

In order to handle round-off error in the acceptance criteria, such as (18) and (20), we relax those slightly based on the machine precision ϵ_{mach}. For example, (20) is replaced in the code by

$$\varphi_{\mu_j}(x_{k,l}) - \varphi_{\mu_j}(x_k) - 10\epsilon_{\text{mach}}|\varphi_{\mu_j}(x_k)| \leq \eta_\varphi \alpha_{k,l} \nabla\varphi_{\mu_j}(x_k)^T d_k^x.$$

4. Numerical Results

In the following sections we examine the practical behavior of the algorithm proposed in this paper. Our implementation, called IPOPT, is written in Fortran 77 and available as open source[3]. The numerical results have been obtained on a PC with a 1.66 GHz Pentium IV microprocessor and 1 GB of memory running RedHat Linux 9.0. The executables were generated with the Intel Fortran compiler version 7.1, using the flags "`-O3 -mp -pc64`". The machine precision is $\epsilon_{\text{mach}} \approx 10^{-16}$. The source code for the required BLAS and LAPACK routines have been obtained from `www.netlib.org` and compiled with the rest of the code.

[3] The source code for IPOPT is available at `http://www.coin-or.org/Ipopt`. In addition, readily available BLAS and LAPACK routines as well as certain subroutines from the Harwell library are required to compile the IPOPT executable.

For the numerical comparison we use the CUTEr test set [18] (as of Jan 1, 2004). Here, problems with general inequality constraints of the form "$d_L \leq d(x) \leq d_U$" are reformulated into the formulation (1) by adding slack variables $d_L \leq s \leq d_U$ and replacing the inequality constraint by "$d(x) - s = 0$." The initial point for the slack variables is chosen as $s_0 = d(\tilde{x}_0)$, where $\tilde{x}_0$ is the starting point given by CUTEr for the original problem formulation.

The test problems initially used in our experiments were all 979 problems with analytical twice continuously differentiable functions that have at least as many free variables as equality constraints, after the reformulation of the general inequality constraints. For problems with variable size we used the default size, except for 46 cases where we decreased the number of variables in order to allow a solution within the given time limit[4]. The problems vary in size from $n = 2$ to 125,050 variables and $m = 0$ to 125,025 constraints (after the introduction of slack variables).

IPOPTwas run for the test set using the default options and a termination tolerance of $\epsilon_{\text{tol}} = 10^{-8}$ with a CPU time limit of 1 hour and an iteration limit of 3000. (The iteration count includes the iterations in the restoration phase algorithms.) Based on those results, we removed 11 problems from the test set, because they appeared unbounded below [5]. The problems `S365` and `S365MOD` were excluded because the constraint Jacobian could not be evaluated at the starting point. In addition, `VANDERM4` was removed since at the initial point $\|c(x_0)\| \approx 10^{63}$, and numerical problems occurred. Finally, we excluded 11 problems on which IPOPT with default options terminated at a point $\tilde{x}_*$ satisfying the termination criterion for the feasibility restoration phase problem (29) (for the tolerance $\epsilon_{\text{tol}} = 10^{-8}$) with $\|c(\tilde{x}_*)\|_1 > \sqrt{\epsilon_{\text{tol}}}$, and for which also the optimization codes KNITRO and LOQO (see Section 4.2 below) both failed. These problems might truly be infeasible[6]. We note that IPOPT was able to converge to a point satisfying the convergence tolerance for the restoration phase problem within the given iteration limit, therefore producing a user message indicating that the problem seems locally infeasible, whereas the other methods (except in two cases) exceeded the iteration limit.

Of the remaining 954 problems, which are those used in the comparisons in the next sections, IPOPT was able to solve 895 problems. This corresponds to a success rate of 93.8%. In 7 cases, it failed to converge within the time limit[7], and in 24 cases the iteration limit was exceeded. Furthermore, IPOPT aborted in 3 problems because it reverted to the restoration phase when the constraint violation was already below the termina-

[4] The problems with altered problem size are `CATENARY, CHARDIS1, CONT5-QP, CONT6-QQ, CVXQP1, CVXQP2, CVXQP3, DRCAV1LQ, DRCAV2LQ, DRCAV3LQ, DRCAVTY3, EG3, EIGENA, EIGENALS, EIGENB, EIGENB2, EIGENBCO, EIGENBLS, EIGENC, EIGENC2, EIGENCCO, EIGENCLS, FLOSP2HH, FLOSP2HL, FLOSP2HM, FMINSURF, GAUSSELM, HARKERP2, LUBRIF, LUBRIFC, NCVXQP[1-9], NONCVXU2, NONMSQRT, POWER, SCURLY30, SPARSINE, SPARSQUR.`

[5] IPOPT failed to converge and was able to produce iterates with very small constraint violation and at the same time very large negative values of the objective function for the following problems: `FLETCBV3, FLETCHBV, INDEF, LUKVLE2, LUKVLE4, LUKVLI2, LUKVLI4, MESH, RAYBENDL, RAYBENDS, STATIC3.`

[6] Those problems were: `CONT6-QQ, DRCAVTY3, FLOSP2HH, FLOSP2HL, FLOSP2HM, HIMMELBD, JUNKTURN, LUBRIF, LUBRIFC, MODEL, WOODSNE.`

[7] The sizes of those problems could not be altered; except for the problems `LUKVLE15` and `LUKVLI10`, the size of which we left unchanged because the time limit was not exceeded for the other problems in the `LUKVL*` family.

tion tolerance, and in 21 problems because the restoration phase algorithm encountered points where the infeasibility was below the convergence tolerance, but the point was not acceptable to the (regular) filter. Finally, in 3 cases IPOPT converged to a stationary point for the infeasibility (but at least one of the codes LOQO and KNITRO was able to solve the problem), and in one case the evaluation of the constraints was repeatedly unsuccessful (producing the IEEE numbers `Inf` or `NaN`).

For the comparisons in the next sections we make use of the Dolan-Moré performance profiles [9]. Given a test set $\mathcal{P}$ containing n_p problems, and n_s runs (e.g., obtained with different solver options) for each problem, these profiles provide a way to graphically present the comparison of quantities $t_{p,s}$ (such as number of iterations or required CPU time) obtained for each problem p and each option s. For this, the performance ratio for a problem p and option s is defined as

$$r_{p,s} := \frac{t_{p,s}}{\min\left\{t_{p,s} : 1 \le s \le n_s\right\}}. \tag{37}$$

If the option s for problem p leads to a failure, we define $r_{p,s} := \infty$. Then,

$$\rho_s(\tau) := \frac{1}{n_p}\text{card}\left\{p \in \mathcal{P} : r_{p,s} \le \tau\right\}$$

is the fraction of the test problems that were solved by the option s within a factor $\tau \ge 1$ of the performance obtained by the best option. The performance plots present ρ_s for each option s as a function of τ; in this paper we use a logarithmic scale for the τ-axis.

Since the considered optimization methods only try to find local solutions of the problems, it can easily happen that two different options converge to different local solutions. In an attempt to avoid comparisons of runs to different local solutions, we exclude those problems for which the final values of the objective functions $f(x_*^1), \ldots, f(x_*^{n_s})$ were not close, that is we discard those problems from a performance plot for which

$$\frac{f_*^{\max} - f_*^{\min}}{1 + \max\{|f_*^{\min}|, |f_*^{\max}|\}} > 10^{-1}, \tag{38}$$

where $f_*^{\max} = \max\{f(x_*^1), \ldots, f(x_*^{n_s})\}$ and $f_*^{\min} = \min\{f(x_*^1), \ldots, f(x_*^{n_s})\}$, with the objective functions values of unsuccessful runs omitted.

4.1. Comparison of Different Line-Search Options

In this section we examine the practical performance of the proposed filter method in comparison with an approach based on the exact penalty function

$$\phi_\nu(x) = \varphi_{\mu_j}(x) + \nu\|c(x)\|. \tag{39}$$

The update rule and step acceptance criteria chosen for the comparison in this paper has been proposed recently by Waltz et. al. in [27] as part of a hybrid trust region and line-search interior-point method. We chose this option since the algorithm in [27] is in many aspects similar to the method proposed here, and since its practical behavior seems promising (in particular, it performs considerably better than the penalty function

approach used in our earlier comparison [24]). In the following we only briefly state the algorithm; its motivation can be found in [27]. We should point out, however, that the algorithm proposed in [27] is more complex and, in particular, reverts, under certain circumstances, to a trust region approach, ensuring global convergence (in contrast to the penalty function option using only a backtracking line-search procedure, see [25]).

For the penalty function based option, the search direction is computed from (13) in an iteration k, and the maximum step sizes are obtained from (15). After this, the penalty parameter is updated according to the formula

$$\nu_k := \begin{cases} \nu_{k-1} & \text{if } \nu_{k-1} \geq \nu_k^+ \\ \nu_k^+ + 1 & \text{otherwise} \end{cases},$$

where

$$\nu_k^+ = \frac{\nabla\varphi_{\mu_j}(x_k)^T d_k^x + \frac{\varsigma_k}{2}(d_k^x)^T(W_k + \Sigma_k + \delta_w I)d_k^x}{(1-\rho)\|c(x_k)\|},$$

for $\rho \in (0,1)$, with $\rho = 0.1$ in our implementation. The scalar ς_k is set to one if $(d_k^x)^T(W_k + \Sigma_k + \delta_w I)d_k^x > 0$, and zero otherwise. After this, a backtracking line-search procedure with $\alpha_{k,l} = 2^{-l}\alpha_k^{\max}$ is performed. For each trial step size $\alpha_{k,l}$ the predicted reduction of the merit function is computed as

$$\begin{aligned}\mathrm{pred}_k(\alpha_{k,l}d_k^x) = -\alpha_{k,l}\nabla\varphi_{\mu_j}(x_k)^T d_k^x - \alpha_{k,l}^2\frac{\varsigma_k}{2}(d_k^x)^T\,(W_k + \Sigma_k + \delta_w I)\,d_k^x + \\ \nu_k\left(\|c(x_k)\| - \|c(x_k) + \alpha_{k,l}\nabla c(x_k)^T d_k^x\|\right)\end{aligned}$$

and compared with the actual reduction

$$\mathrm{ared}_k(\alpha_{k,l}d_k^x) = \phi_{\nu_k}(x_k) - \phi_{\nu_k}(x_k + \alpha_{k,l}d_k^x).$$

If

$$\mathrm{ared}_k(\alpha_{k,l}d_k^x) < \eta\ \mathrm{pred}_k(\alpha_{k,l}d_k^x), \tag{40}$$

for a constant $\eta \in (0, \frac{1}{2})$, then the trial step size $\alpha_k = \alpha_{k,l}$ is accepted and the iterates are updated according to (14). Otherwise a shorter trial step size is tried. It can be shown that d_k^x is a descent direction for ϕ_{ν_k}, so that eventually a sufficiently small trial step size is accepted. At the beginning of the optimization and after each decrease of the barrier parameter μ_j, the penalty parameter ν_k is set to 10^{-6}.

In order to achieve a fair comparison between the filter method and this approach, all comparable constants (such as η) are set to the same values, so that the methods behave identically on problems without equality constraints. In addition, the details described in Section 3 are identical (including a second-order correction for the merit function method), unless they pertain specifically to the filter method, such as the restoration phase. In particular, the Hessian correction scheme described in Section 3.1 is also used for the line-search algorithm using (39), with the exception that δ_c is always kept at zero to ensure that the generated search direction d_k^x is always a descent direction for the exact penalty function. As a consequence, this line-search option aborts, when a point with

rank-deficient constraint Jacobian is reached and no search direction can be computed, because the linear system (13) is singular.

We first compare the default filter method (labeled "`Filter (default)`") with the penalty function approach just described ("`Penalty Function`") in terms of iteration counts. However, since the default filter procedure includes a few heuristics that are not used in the penalty function approach, we also include the option "`Filter (no heuristics)`," for which the heuristics described in Section 3.2 have been disabled, and for which $\delta_c = 0$ in (13) all the time. Finally, we also include the option "`Full Step`", for which the backtracking line-search procedure has been disabled, i.e., in every iteration the step size $\alpha_k = \alpha_k^{\max}$ is chosen.

The performance plot presented in Figure 1 summarizes the comparison on 932 problems (22 were omitted because their final objective function values were different, see (38)). As one can see, the filter option is indeed more robust than the penalty function method, even when the heuristics are disabled. We can also conclude that the introduction of the heuristics in Section 3.2 and the relaxation $\delta_c \geq 0$ in (13) increases the robustness of the method. Finally, the comparison with the "`Full Step`" option seems to indicate that the safeguards of the filter and penalty function method, which have been introduced to guarantee global convergence, do not interfere, in an overall sense, with the efficiency of Newton's method. Note that the "`Full Step`" option still does relatively well in terms of robustness (for 86.1% of the considered problems the algorithm stopped at a point satisfying the termination criterion); this might indicate

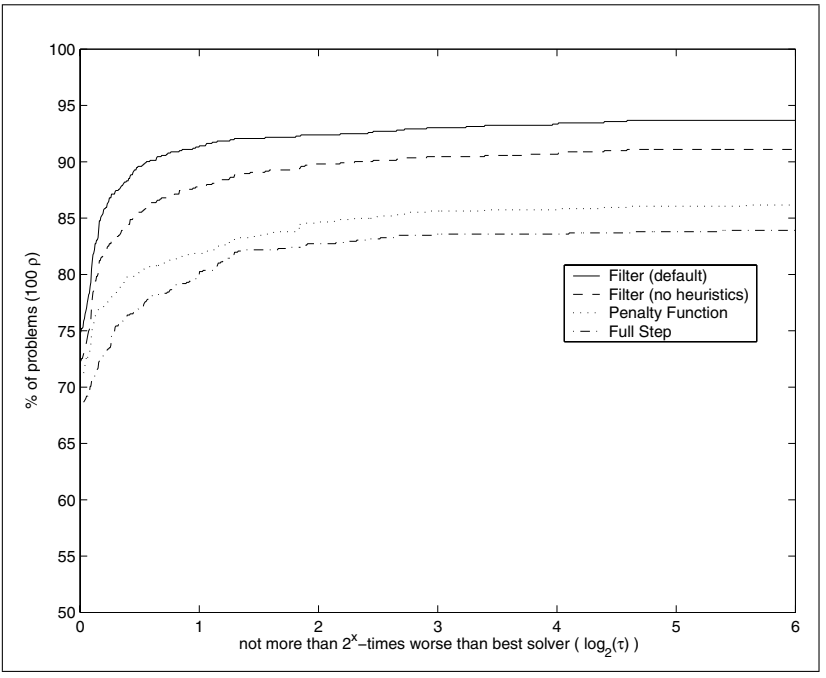

Fig. 1. Comparing iteration count for different line-search options

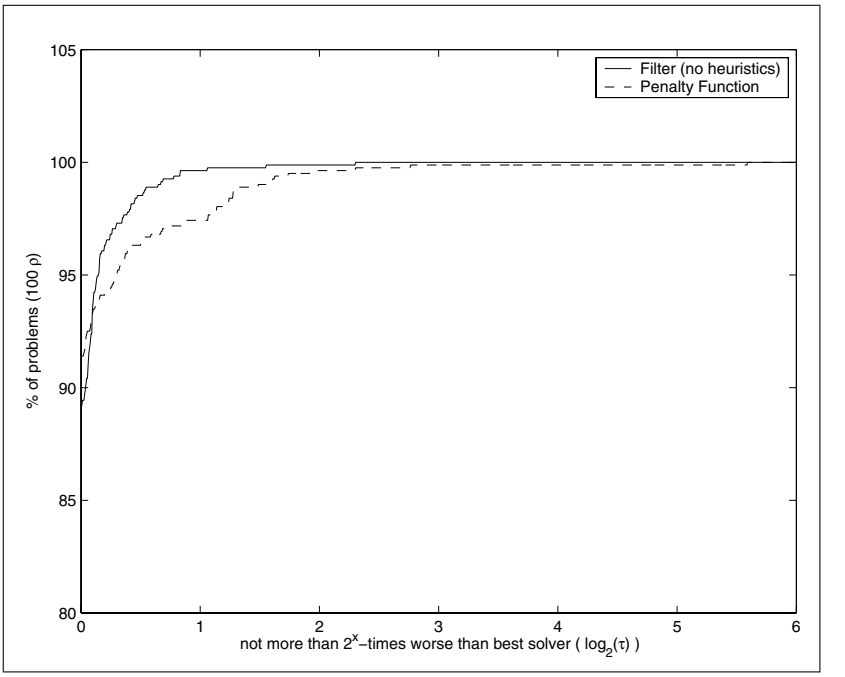

Fig. 2. Comparing iteration count for problems solved by "`Filter (no heuristics)`" and "`Penalty Function`"

that in many cases Newton's method does not require a safeguarding scheme (note that second derivatives are used in the computation of the search directions), or alternatively, that many problems in the test set are not very difficult.

On the other hand, the different options do not seem to differ very much in terms of efficiency. If we compare the number of iterations for the "`Filter (no heuristics)`" and "`Penalty Function`" only for those 814 problems, in which both options were able to find a solution with comparable final objective function values, then the performance plots turn out to be very similar, see Figure 2 (note that the range of the vertical axis starts at 80%). The filter option seems to be only slightly more efficient for those problems.

4.2. Comparison with Other Interior-Point Codes

In this section we present a comparison of IPOPT with the optimization codes KNITRO [3, 28] (version 3.1.1) and LOQO [23] (version 6.06), both well regarded and recognized software packages for large-scale nonlinear optimization. Like IPOPT, they are based on interior-point approaches. Tables with detailed results for every test problem and each solver can be downloaded from the first author's home page[8].

[8] `http://www.research.ibm.com/people/a/andreasw`

The comparison presented here is not meant to be a rigorous assessment of the performance of these three algorithms, as this would require very careful handling of subtle details such as comparable termination criteria etc, and would be outside the scope of this paper. In addition, all three software packages are continuously being improved, so that a comparison might quickly be out of date. The main purpose of the comparison here is to give an idea of the relative performance of IPOPT, and to encourage readers to consider IPOPT as a potential candidate when looking for a practical nonlinear optimization code.

All three optimization codes were run with their default options on the 954 problems of our test set on the same machine as used to obtain the IPOPT results. Again, a CPU time limit of 1 hour and an iteration count limit of 3000 was imposed. The default termination tolerance for KNITRO and LOQO is "10^{-6}," whereas we still chose $\epsilon_{\text{tol}} = 10^{-8}$ for IPOPT. The termination criteria are not directly comparable, for example due to different scalings of various entities and different reformulations of the problems, but we believe that on average the chosen termination criterion for IPOPT is tighter than those for the other two codes. We include a run for IPOPT, for which the automatic problem scaling procedure described in Section 3.8 has been disabled, since the other codes do not perform any scaling of the problem statement.

As mentioned earlier, IPOPT in default mode terminated successfully for 895 out of the 954 problems, whereas only 872 could be solved when the scaling was disabled. KNITRO terminated successfully in 829 cases, and LOQO for 847 problems. Figure 3

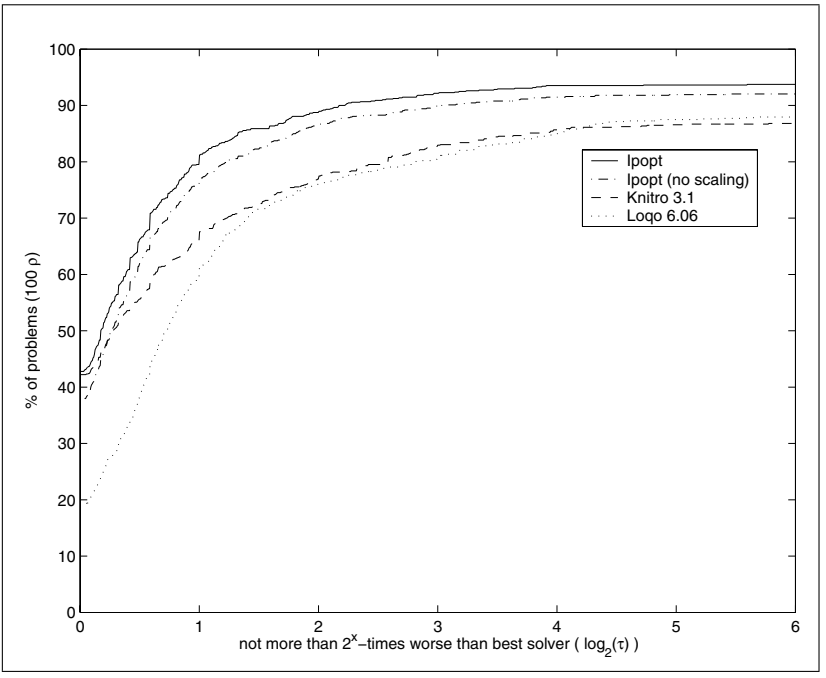

Fig. 3. Comparing solvers (iteration count)

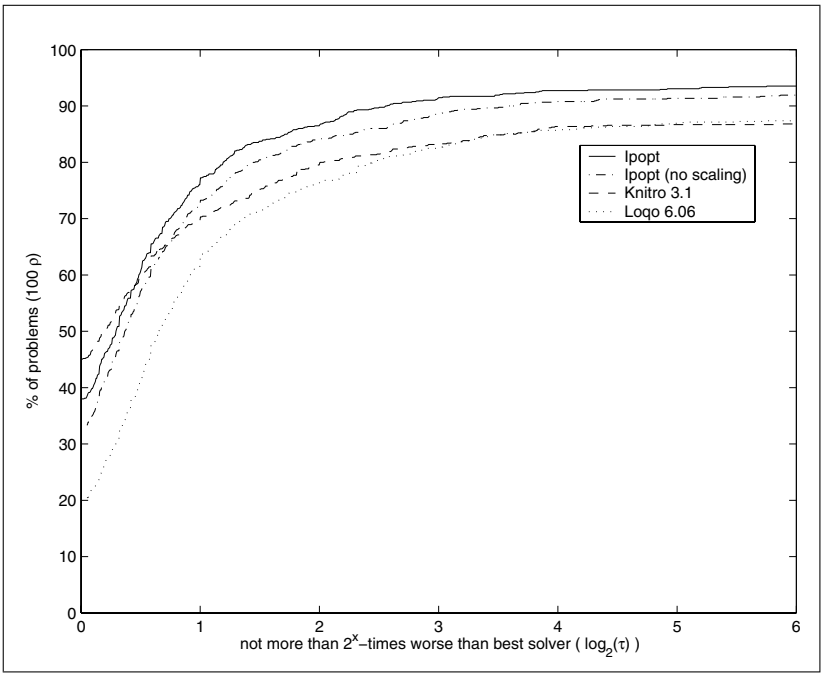

Fig. 4. Comparing solvers (function evaluations)

presents a performance plot for the iteration count, and Figure 4 compares the number of function evaluations[9]. Here, 75 problems were excluded because the final objective function values were too different (see (38)). IPOPT appears to be more efficient in both measures compared to LOQO, and comparable to KNITRO in terms of iteration counts. However, KNITRO is a trust region method, and the computational costs per iteration are usually not comparable; each unsuccessful trial point in KNITRO is counted as one iteration. Looking at Figure 4, KNITRO seems to require overall fewer function evaluations than IPOPT for the given test set. These figures also show that the scaling procedure proposed in Section 3.8 does indeed improve IPOPT's robustness and efficiency.

Finally, Figure 5 presents a comparison of the CPU time[10]. Since the CPU time is measured in 0.01s increments on the machine used for obtaining the results, we excluded the 444 test problems from the graph, for which the CPU time for the fastest solver was less than 0.05s, as well as 48 additional problems with different final objective function values. As can be seen, IPOPT seems to perform well compared to the other solvers.

[9] LOQO appears to compute the function value for each accepted iterate twice, so that a minimum of two function evaluations is observed per iteration. To correct for this, the function evaluation count for LOQO has been decreased by the number of iterations for the performance plots.

[10] Like IPOPT, KNITRO is written in Fortran and has been compiled with the same compiler and compiler options. LOQO is written C, and we used the default Linux library available at the LOQO website.

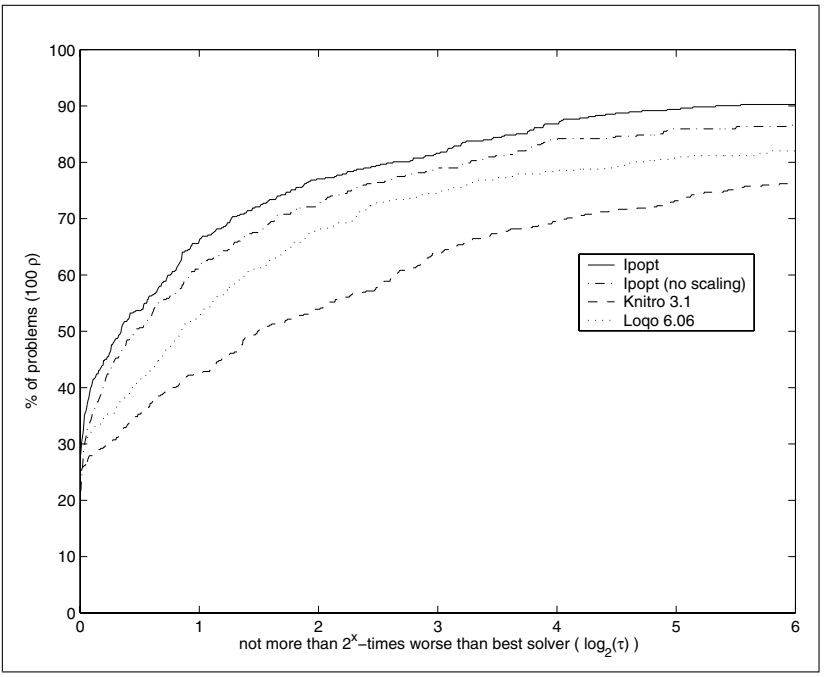

Fig. 5. Comparing solvers (CPU time)

5. Conclusions

We presented a detailed description of an interior-point nonlinear programming algorithm based on a filter line search. Attention has been paid to a number of algorithmic features including the incorporation of second-order corrections and an efficient and robust feasibility restoration phase. Further implementation details include inertia correction of the linear system that determines the search direction, treatment of unbounded solution sets, two acceleration heuristics, as well as automatic problem scaling. The resulting algorithm is implemented in the IPOPT open source software package. The performance of the code has been demonstrated with a detailed numerical study based on 954 problems from the CUTEr test set. An evaluation of several line-search options has been presented, indicating increased robustness due to the filter approach. Also, a comparison has been provided with the LOQO and KNITRO codes. These results demonstrate favorable performance of IPOPT.

Acknowledgements. The authors would like to thank Richard Waltz and Jorge Nocedal, as well as Hande Benson and Robert Vanderbei for their help and providing a copy of their optimization codes KNITRO and LOQO, respectively. We further thank Arvind Raghunathan for insightful comments on different aspects of the algorithm, Carl Laird for his help in obtaining the numerical results, and Dominique Orban for support on CUTEr issues. We are also very grateful to Andrew Conn and Jorge Nocedal, as well as two anonymous referees, whose comments on the manuscript greatly helped to improve the exposition of the material.

References

1. Benson, H. Y., Shanno, D. F., Vanderbei, R. J.: Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions. Computational Optimization and Applications, **23** (2), 257–272 (2002)
2. Byrd, R. H., Gilbert, J. Ch., Nocedal, J.: A trust region method based on interior point techniques for nonlinear programming. Mathematical Programming, **89**, 149–185 (2000)
3. Byrd, R. H., Hribar, M. E., Nocedal, J.: An interior point algorithm for large-scale nonlinear programming. SIAM Journal on Optimization, **9** (4), 877–900 (1999)
4. Byrd, R. H., Liu, G., Nocedal, J.: On the local behavior of an interior point method for nonlinear programming. In: Griffiths, D. F., Higham, D. J. (eds), Numerical Analysis 1997, pages 37–56. Addison–Wesley Longman, Reading, MA, USA, 1997
5. Chamberlain, R. M., Lemarechal, C., Pedersen, H. C., Powell, M. J. D.: The watchdog technique for forcing convergence in algorithms for constrained optimization. Mathematical Programming Study, **16**, 1–17 (1982)
6. Conn, A. R., Gould, N. I. M., Toint, Ph. L.: LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A). Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992
7. Conn, A. R., Gould, N. I. M., Toint, Ph. L.: Trust-Region Methods. SIAM, Philadelphia, PA, USA, 2000
8. Conn, A. R., Gould, N.I.M., Orban, D., Toint, Ph. L.: A primal-dual trust-region algorithm for non-convex nonlinear programming. Mathematical Programming, **87** (2), 215–249 (2000)
9. Dolan, E. D., Moré, J. J.: Benchmarking optimization software with performance profiles. Mathematical Programming, **91** (2), 201–213 (2002)
10. El-Bakry, A. S., Tapia, R. A., Tsuchiya, T., Zhang, Y.: On the formulation and theory of the Newton interior-point method for nonlinear programming. Journal of Optimization Theory and Application, **89** (3), 507–541 (1996)
11. Fiacco, A. V., McCormick, G. P.: Nonlinear Programming: Sequential Unconstrained Minimization Techniques. John Wiley, New York, USA, 1968 Reprinted by SIAM Publications, 1990.
12. Fletcher, R.: Practical Methods of Optimization. John Wiley and Sons, New York, USA, second edition, 1987
13. Fletcher, R., Gould, N. I. M., Leyffer, S., Toint, Ph. L., Wächter, A.: Global convergence of a trust-region SQP-filter algorithms for general nonlinear programming. SIAM Journal on Optimization, **13** (3), 635–659 (2002)
14. Fletcher, R., Leyffer, S.: Nonlinear programming without a penalty function. Mathematical Programming, **91** (2), 239–269 (2002)
15. Fletcher, R., Leyffer, S., Toint, Ph. L.: On the global convergence of a filter-SQP algorithm. SIAM Journal on Optimization, **13** (1), 44–59 (2002)
16. Forsgren, A., Gill, P. E., Wright, M. H.: Interior methods for nonlinear optimization. SIAM Review, **44** (4), 525–597 (2002)
17. Gould, N. I. M., Orban, D., Sartenaer, A., Toint, Ph. L.: Superlinear convergence of primal-dual interior point algorithms for nonlinear programming. SIAM Journal on Optimization, **11** (4), 974–1002 (2001)
18. Gould, N. I. M., Orban, D., Toint, Ph. L.: CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited. Technical Report TR/PA/01/04, CERFACS, Toulouse, France, 2001
19. Harwell Subroutine Library, AEA Technology, Harwell, Oxfordshire, England. A catalogue of subroutines (HSL 2000), 2002
20. Nocedal, J., Wright, S.: Numerical Optimization. Springer, New York, NY, USA, 1999
21. Tits, A. L., Wächter, A., Bakhtiari, S., Urban, T. J., Lawrence, C. T.: 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)
22. Ulbrich, M., Ulbrich, S., Vicente, L. N.: A globally convergent primal-dual interior-point filter method for nonlinear programming. Mathematical Programming, **100** (2), 379–410 (2004)
23. Vanderbei, R. J., Shanno, D. F.: An interior-point algorithm for nonconvex nonlinear programming. Computational Optimization and Applications, **13**, 231–252 (1999)
24. Wächter, A.: An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, January 2002
25. Wächter, A., Biegler, L. T.: Failure of global convergence for a class of interior point methods for nonlinear programming. Mathematical Programming, **88** (2), 565–574 (2000)
26. Wächter, A., Biegler, L. T.: Line search filter methods for nonlinear programming: Motivation and global convergence. Technical Report RC 23036, IBM T.J. Watson Research Center, Yorktown Heights, USA, 2001; revised 2004. To appear in SIAM Journal on Optimization.

27. Waltz, R. A., Morales, J. L., Nocedal, J., Orban, D.: An interior algorithm for nonlinear optimization that combines line search and trust region steps. Technical Report OTC 6/2003, Optimization Technology Center, Northwestern University, Evanston, IL, USA. To appear in Mathematical Programming A
28. Waltz, R. A., Nocedal, J.: KNITRO user's manual. Technical Report OTC 2003/05, Optimization Technology Center, Northwestern University, Evanston, IL, USA, April 2003
29. Yamashita, H.: A globally convergent primal-dual interior-point method for constrained optimization. Optimization Methods and Software, **10**, 443–469 (1998)
30. Yamashita, H., Yabe, H., Tanabe, T.: A globally and superlinearly convergent primal-dual interior point trust region method for large scale constrained optimization. Technical report, Mathematical System Institute, Inc., Tokyo, Japan, July 1997. Revised July 1998