A Note on the Implementation of an Interior-Point Algorithm for Nonlinear Optimization with Inexact Step Computations

Frank E. Curtis · Johannes Huber · Olaf Schenk · Andreas Wächter

April 5, 2012

Abstract This paper describes an implementation of an interior-point algorithm for large-scale nonlinear optimization. It is based on the algorithm proposed by Curtis, Schenk, and Wächter [SIAM J. Sci. Comput., 32 (2010), pp. 3447-3475], a method that possesses global convergence guarantees to first-order stationary points with the novel feature that inexact search direction calculations are allowed in order to save computational expense. The implementation follows the proposed algorithm, but includes many practical enhancements, such as functionality to avoid the computation of a normal step during every iteration. The implementation is included in the IPOPT software package paired with an iterative linear system solver and preconditioner provided in PARDISO. Numerical results on a large nonlinear optimization test set and two PDE-constrained optimization problems with control and state constraints are presented to illustrate that the implementation is robust and efficient for large-scale applications.

Keywords large-scale optimization, PDE-constrained optimization, interior-point methods, nonconvex programming, line search, trust regions, inexact linear system solvers, Krylov subspace methods

Mathematics Subject Classification (2000) 49M05 · 49M15 · 49M37 · 65F10 · 65K05 · 65N22 · 90C06 · 90C26 · 90C30 · 90C51 · 90C90

Frank E. Curtis
Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA.
E-mail: frank.e.curtis@lehigh.edu

Johannes Huber
Department of Mathematics and Computer Science, University of Basel, Basel, Switzerland.
E-mail: johannes.huber@unibas.ch

Olaf Schenk
Institute of Computational Science, Università della Svizzera italiana, Lugano, Switzerland.
E-mail: olaf.schenk@usi.ch

Andreas Wächter
Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA.
E-mail: andreas.waechter@northwestern.edu
1 Introduction

The techniques described in this paper are motivated by increased interests in the solution of large-scale nonlinear optimization problems. By large-scale, we refer to classes of problems for which contemporary optimization techniques, including most interior-point methods, have proved to be impractical due to large numbers of variables/constraints and significant fill-in during the factorization of derivative matrices. New computationally efficient strategies are needed if such large-scale problems are to be solved realistically in practical situations.

The main purpose of this paper is to describe a practical implementation, including enhanced algorithmic features, for the algorithm proposed and analyzed in [13]. This algorithm addresses the challenges posed in large-scale nonlinear optimization by employing iterative linear system solvers in place of direct factorization methods when solving the large-scale linear systems involved in an interior-point strategy. Moreover, computational flexibility is greatly increased as inexact search direction calculations are allowed, but controlled sufficiently so that theoretical convergence guarantees are maintained. Our experience has shown that the implementation described in this paper achieves these desirable characteristics.

A prime example of a class of problems for which our techniques may be applicable are those where the constraints involve discretized partial differential equations (PDEs) [4,5,20]. Typical methods for solving these types of problems generally fall into the categories of nonlinear elimination [1,3,23], reduced space [19,21], or full space [6,7,18] techniques. The algorithm discussed in this paper fits into the category of full-space methods, but is unique from many previously proposed approaches in its ability to attain strong theoretical convergence guarantees with great computational flexibility.

We describe our implementation in the context of the generic problem

\[
\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad c_E(x) = 0 \quad \text{and} \quad c_I(x) \geq 0,
\]

where the objective \( f : \mathbb{R}^n \to \mathbb{R} \), equality constraints \( c_E : \mathbb{R}^n \to \mathbb{R}^p \), and inequality constraints \( c_I : \mathbb{R}^n \to \mathbb{R}^q \) are assumed to be sufficiently smooth (e.g., \( C^2 \)). If problem (1.1) is infeasible, however, then our algorithm is designed to return a stationary point for the unconstrained problem

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2} \|c_E(x)\|^2_2 + \frac{1}{2} \max\{-c_I(x), 0\}^2
\]

(1.2)

as a certificate of infeasibility. A solution to (1.2) that does not satisfy the constraints of problem (1.1) is known as an infeasible stationary point.

While the algorithm described in this paper is similar to the one presented in [13], we present its implementation here in more detail. In addition, there are a number of notable differences. The primary difference is the strategy we describe for switching between two step computation methods. Our goal is to improve the overall efficiency of the algorithm while still ensuring the convergence guarantees provided in [13]. The other main differences include refinements of the termination tests for the iterative linear system solver and Hessian modification strategy, as well as a new adaptive refinement strategy in the preconditioner computation. Due to space limitations, some of the motivations behind our techniques are omitted. A more comprehensive development can be found in the full-length version [10].
The focus of our numerical experiments is the performance of the nonlinear optimization algorithm when an iterative solver is used for the inexact solution of the arising linear systems. We note, however, that in practice it is also extremely important to employ effective preconditioners. For example, in the context of PDE-constrained optimization, advantages can be gained by exploiting spectral properties of discrete differential operators. In our study, we obtain very encouraging results using the general-purpose preconditioner implemented in PARDISO; see §3.1.

Notation. All norms are considered $\ell_2$. We drop function dependencies once they are defined and, when applicable, apply iteration number information to the function itself; i.e., by $c_k$, we mean $c(x_k)$.

2 Algorithm Description

We motivate and describe our implemented algorithm in this section. The method is based on the series of inexact SQP, Newton, and interior-point algorithms that have been proposed and analyzed in [8,9,12,13], though the majority relates to the latest enhancements in [13]. We begin by describing the basic interior-point framework of the approach, and then discuss the major computational component of the algorithm, namely, the search direction calculation. Further specifications and details of our software implementation are provided in §3 and §4.

It is important to note that, in this section, we consider scaled derivatives corresponding to the slack variables for the inequality constraints; see $\gamma(z;\mu)$, $A(z)$, and $W(z,\lambda;\mu)$ throughout this section. This results in scaled sets of equations for computing the primal-dual step; see [10] for details. In particular, we denote $\Sigma \approx S$ as a diagonal scaling matrix that depends on the values of the slack variables $s$.

2.1 Interior-Point Framework

The framework of the algorithm is a classical interior-point strategy. Let $z = (x, s)$ be the primal variables, where $s \in \mathbb{R}^q$ is a vector of slack variables, and let

$$\varphi(z;\mu) := f(x) - \mu \sum_{i=1}^{q} \ln s^{(i)} \quad \text{and} \quad c(z) := \begin{bmatrix} c_E(x) \\ c_I(x) - s \end{bmatrix}.$$  

For a sequence of barrier parameters $\mu \downarrow 0$, problem (1.1) is solved through the solution of a sequence of barrier subproblems of the form

$$\min_{z} \varphi(z;\mu) \quad \text{s.t.} \quad c(z) = 0. \tag{2.1}$$

If $f$, $c_E$, and $c_I$ are continuously differentiable, then first-order Karush-Kuhn-Tucker (KKT) optimality conditions for (2.1) are

$$\begin{bmatrix} \gamma(z;\mu) + A(z)^T \lambda \\ c(z) \end{bmatrix} = 0 \tag{2.2}$$

with $s \geq 0$, where $\lambda \in \mathbb{R}^{p+q}$ are Lagrange multipliers, $e \in \mathbb{R}^q$ is a vector of ones,

$$\gamma(z;\mu) := \begin{bmatrix} \nabla f(x) \\ -\mu S^{-1} \Sigma e \end{bmatrix}, \quad \text{and} \quad A(z) := \begin{bmatrix} \nabla c_E(x)^T & 0 \\ \nabla c_I(x)^T & -\Sigma \end{bmatrix}.$$
In situations where (1.1) is infeasible, (2.1) has no solution, so the algorithm transitions from solving (2.1) to solving (1.2). The latter problem has the KKT conditions $A(z)^T c(z) = 0, s \geq 0,$ and $c_T(x) - s \leq 0$. In fact, the algorithm maintains $s \geq 0$ and $c_T(x) - s \leq 0$ during each iteration by increasing $s$ when necessary. Thus, convergence to a solution of the barrier subproblem (2.1) or an infeasible stationary point of (1.1) is achieved once (2.2) or $A(z)^T c(z) = 0,$ respectively, is satisfied.

At an iterate $(z_k, \lambda_k)$, the algorithm computes a primal-dual search direction $(d_k, \delta_k)$ satisfying appropriate conditions for guaranteeing global convergence; see §2.2. Given such a direction, we compute the scaled direction $e_k d_k := (d_k^x, \Sigma_k d_k^s)$ along which a line search is performed. The line search involves two conditions. First, to maintain positivity of the slacks, a stepsize $\alpha_{k \max} \in (0, 1]$ satisfying

$$s_k + \alpha_{k \max}^\max \Sigma_k d_k^s \geq (1 - \eta_1) s_k$$

is determined for a constant $\eta_1 \in (0, 1)$. We use $\eta_1 = \max\{0.99, 1 - \mu\}$ in our implementation. The algorithm then backtracks from this value to compute $\alpha_k \in (0, \alpha_{k \max}^\max]$ yielding sufficient decrease in the penalty function

$$\phi(z_k; \mu, \pi) := \varphi(z_k; \mu) + \pi \|c(z_k)\|,$$

where $\pi > 0$ is a penalty parameter. The condition we enforce is

$$\phi(z_k + \alpha_k d_k; \mu, \pi) \leq \phi(z_k; \mu, \pi) - \eta_2 \alpha_k \Delta m_k(d_k; \mu, \pi)$$

where $\eta_2 \in (0, 1)$ is a constant (we choose $\eta_2 = 10^{-8}$), and where $\Delta m_k(d_k; \mu, \pi)$ relates to the directional derivative of $\phi$ along the computed search direction. We define $\Delta m_k(d_k; \mu, \pi)$ in equation (2.9) in §2.2. In the dual space, we update $\lambda_{k+1} \leftarrow \lambda_k + \beta_k \delta_k$ where $\beta_k$ is the smallest value in $[\alpha_k, 1]$ satisfying

$$\|\gamma_k + A_k^T \lambda_{k+1}\| \leq \|\gamma_k + A_k^T (\lambda_k + \delta_k)\|.$$ 

Our complete interior-point framework is presented as Algorithm [1]. In the algorithm, we refer to two search direction computation variants, Algorithms [2] and [3] presented in §2.2. These methods include mechanisms for computing $(d_k, \delta_k)$ and updating the penalty parameter $\pi$. The termination criteria for the original problem (1.1) and the barrier problem (2.1), the choice of the initial point, and the updating rule for the barrier parameter $\mu$ are identical with those in [28].

2.2 Search Direction Computation

The main computational component of Algorithm [1] is the primal-dual search direction calculation (step 4). We describe two approaches for computing this direction, presented as Algorithms [2] and [3]. The first approach is simpler, but global convergence for Algorithm [1] is only guaranteed with this method if an infinite subsequence of iterations involve (scaled) constraint Jacobians $\{A_k\}$ that have full row rank and singular values bounded away from zero. Otherwise, the second approach must be employed to ensure convergence. The two algorithms have many common features, and we start by discussing the components present in both techniques. (In §3.1 we discuss our mechanism for having the algorithm dynamically choose between these two approaches during each iteration of Algorithm [1].)
Algorithm 1  Interior-Point Framework

1: (Initialization) Choose line search parameters \( \eta_1, \eta_2 \in (0, 1) \), an initial barrier parameter \( \mu > 0 \), and an initial penalty parameter \( \pi > 0 \). Initialize \((x_0, s_0, \lambda_0)\) so that the slack variables satisfy \( s_0 > 0 \) and \( s_0 \geq c_i(x_0) \). Set \( k = 0 \).

2: (Tests for convergence) If convergence criteria for \((1.1)\) are satisfied, then terminate and return \( x_k \) as an optimal solution. Else, if convergence criteria for \((1.2)\) are satisfied and \( x_k \) is infeasible for \((1.1)\), then terminate and return \( x_k \) as an infeasible stationary point.

3: (Barrier parameter update) If convergence criteria for \((1.1)\) are satisfied, then decrease the barrier parameter \( \mu > 0 \), reset \( \pi > 0 \), and go to step 2.

4: (Search direction computation) Compute \((d_k, \delta_k)\) and update \( \pi \) by Algorithm 2 or Algorithm 3. Set the search direction as \( d_k \leftarrow (d_k^\top, \Sigma_k \delta_k^\top) \).

5: (Line search) If \( d_k = 0 \), then \( \alpha_k \rightarrow 1 \). Else, let \( \alpha_k^\max \) be the largest value in \((0, 1)\) satisfying \((2.3)\) and let \( l \) be the smallest value in \( \mathbb{N}_0 \) such that \( \alpha_k \rightarrow -2^{-l} \alpha_k^\max \) satisfies \((2.5)\).

6: (Iterate update) Set \( z_{k+1} \leftarrow z_k + \alpha_k d_k, s_{k+1} \leftarrow \max\{s_{k+1}, c_i(x_{k+1})\} \), update \( \lambda_{k+1} \) according to \((2.6)\), set \( k \leftarrow k + 1 \), and go to step 3.

The search direction computation is based on Newton’s method applied to the KKT conditions of problem \((2.1)\). Defining the scaled Hessian matrix

\[
W(z, \lambda; \mu) := \begin{bmatrix}
\nabla^2_{xx} f & 0 \\
0 & \lambda \Sigma \Xi \Sigma^T
\end{bmatrix} + \sum_{i=1}^p \lambda_i \left( \begin{bmatrix}
\nabla^2_{xx} c_i^T & 0 \\
0 & 0
\end{bmatrix} + \sum_{i=1}^q \lambda_i \left( \begin{bmatrix}
\nabla^2_{xx} c_i^T & 0 \\
0 & 0
\end{bmatrix}
\right)
\right),
\]

(2.7)
a Newton iteration for \((2.8)\) is defined by the linear system

\[
\begin{bmatrix}
W_k & A_k^T \\
A_k & 0
\end{bmatrix}
\begin{bmatrix}
d_k \\
\delta_k
\end{bmatrix} = -\begin{bmatrix}
\gamma_k + A_k^T \lambda_k \\
c_k
\end{bmatrix}.
\]

(2.8)

In \((2.7)\), \( \Xi \) is an approximate Hessian for the barrier term; see \((3.2)\).

The central issue that must be confronted when applying Newton’s method for large-scale applications is that exact solutions of \((2.8)\) may be computationally expensive to obtain. Therefore, our major concern is how an iterative linear system solver can be employed for solving \((2.8)\) in such a way that inexact solutions are allowed, yet global convergence of the algorithm is guaranteed. This issue was the inspiration for all of the algorithms proposed in \([8,9,12,13]\).

Algorithms 2 and 3 each outline a series of termination tests for an iterative solver applied to \((2.8)\) that state conditions under which an inexact solution \((d_k, \delta_k)\) can be considered an acceptable direction for step 4 in Algorithm 1. Of central importance in these termination tests are the residual vectors

\[
\rho_k(d_k, \delta_k) := \gamma_k + W_k d_k + A_k^T (\lambda_k + \delta_k) \quad \text{and} \quad r_k(d_k) := c_k + A_k d_k,
\]

as well as the primal-dual relative residual

\[
\psi_k(d_k, \delta_k) := \begin{bmatrix}
\rho_k(d_k, \delta_k) \\
r_k(d_k)
\end{bmatrix} \begin{bmatrix}
\gamma_k + A_k^T \lambda_k \\
c_k
\end{bmatrix}.
\]

(2.2)

To promote fast convergence, \( \psi_k \) should be small \([14]\). Thus, our implementation aims to compute steps for which this relative residual is below a desired threshold, but appropriate fallbacks are in place in case this is difficult to achieve.

For convex problems, the algorithm can focus exclusively on \( \psi_k \), terminating the calculation of \((d_k, \delta_k)\) whenever this value is below a threshold \([14]\). For non-convex problems, however, the priority is to find solutions to \((2.2)\) that correspond
to minimizers, not saddle points or maximizers. The methods developed in 5,8,9,12 therefore include additional conditions and procedures that aid the algorithms in converging toward minimizers of \((2.1)\). These additional conditions involve a local model of the penalty function \(\phi(z; \mu, \pi)\) at \(z_k\), denoted as

\[
m_k(d; \mu, \pi) := \varphi_k + \gamma_k^T d + \pi \|e_k + A_k d\|,
\]

and the reduction in this model yielded by \(d_k\), which is defined as

\[
\Delta m_k(d_k; \mu, \pi) := m_k(0; \mu, \pi) - m_k(d_k; \mu, \pi).
\]  

It can be shown that \(D\varphi_k(\bar{d}_k; \mu, \pi) \leq -\Delta m_k(d_k; \mu, \pi)\), where \(D\varphi_k(\bar{d}_k; \mu, \pi)\) is the directional derivative of \(\phi(\cdot; \mu, \pi)\) at \(z_k\) along \(d_k\); see 13. To ensure that \(d_k\) is always a descent direction for \(\phi(\cdot; \mu, \pi)\) at \(z_k\), the termination tests require that \(\Delta m_k(d_k; \mu, \pi)\) is sufficiently large, potentially after an increase of \(\pi\).

We now present our first step computation approach, Algorithm 2. The algorithm contains three termination tests that are similar in form to those contained in Algorithm 3 later on. In 8,9,12,13, these tests are called sufficient merit function approximation reduction termination tests (SMART tests, for short) due to the significant role that \(\Delta m_k(d_k; \mu, \pi)\) plays in the theoretical behavior the algorithm. For our numerical experiments, we choose \(J = 100\), \(\kappa_{des} = 10^{-3}\), \(\kappa = 10^{-2}\), \(\epsilon_1 = 0.09\), \(\theta = 10^{-12}\mu\) (where \(\mu\) is the current value of the barrier parameter), \(\zeta = 10^{-4}\), \(\tau = 0.1\), \(\kappa_3 = 10^{-3}\), \(\epsilon_3 = 10^{-8}\), and \(\kappa_M = 10^{-2}\). In particular, we typically require the relative residual to be only less than \(\kappa = 10^{-2}\) or \(\kappa_{des} = \kappa_3 = 10^{-3}\), but the actual tolerance is dictated by the entirety of the termination criteria. The value for \(\xi\) is chosen as described in 3.2.

Our second approach, Algorithm 3, is a replacement for Algorithm 2 when the scaled constraint Jacobian may be ill-conditioned or rank-deficient. In such cases, it is necessary to regularize the step computation since otherwise the calculation may not be well-defined, or at best may lead to long, unproductive search directions.

Algorithm 3 performs this regularization by decomposing the search direction as \(d_k \leftarrow v_k + u_k\), where the normal component \(v_k\) represents a direction toward linearized feasibility and the tangential component \(u_k\) represents a direction toward optimality. The normal component \(v_k\) is defined as an approximate solution to

\[
\min_v \frac{1}{2} \|r_A(v)\|^2 \text{ s.t. } \|v\| \leq \omega \|A_k^T c_k\| \tag{2.11}
\]

for some \(\omega > 0\). The trust region constraint regularizes the computation of the normal step and controls the size of this component even when \(A_k\) loses rank. We initialize \(\omega \leftarrow 100\), but have found it practically beneficial to set \(\omega \leftarrow \min\{10\omega, 10^{20}\}\) if \(\|r_k\| = \omega \|A_k^T c_k\|\) and \(\alpha_k = 1\) at the end of iteration \(k\); see 12.

As with the linear system \((2.8)\), the exact solution of \((2.11)\) is expensive. However, global convergence is guaranteed as long as \(v_k\) is feasible for problem \((2.11)\) and satisfies Cauchy decrease, i.e., \(\|c_k\| - \|r_k(v_k)\| \geq \epsilon_v (\|c_k\| - \|r_k(\bar{\alpha}_k \bar{v}_k)\|)\) for some constant \(\epsilon_v \in (0, 1)\) (we choose \(\epsilon_v = 0.1\)); see 12,18. Here, the vector \(\bar{v}_k := -A_k^T c_k\) is the steepest descent direction for problem \((2.11)\) at \(v = 0\) and the steplength \(\bar{\alpha}_k\) is the minimizer of \(\frac{1}{2} \|c_k + \bar{\alpha}A_k \bar{v}_k\|^2\) over all \(\bar{\alpha} \leq \omega\).

A number of techniques have been developed for the inexact solution of large-scale instances of problem \((2.11)\) with solutions satisfying Cauchy decrease; e.g., see the CG method described in 26. In our software, we have implemented an
Algorithm 2  Inexact Newton Iteration with SMART Tests

1: (Initialization) Choose parameters \( J \in \mathbb{N}_0, n_{\text{tol}} \in (0, 1), \kappa \in (0, 1), \epsilon_1 \in (0, 1), \theta > 0, \zeta > 0, \tau \in (0, 1), \kappa_3 \in (0, 1), \epsilon_3 > 0, \) and \( \kappa_W > 0. \) Choose a diagonal matrix \( D_k > 0. \) Initialize \( j \leftarrow 1 \) and \((d_k, \delta_k) \leftarrow (0, 0).\)

2: (Residual test) If \( j \leq J \) and \( \Psi_k > n_{\text{tol}}, \) then go to step 7.

3: (Termination test 1) If \( \Psi_k \leq \kappa \) and the model reduction condition

\[
\Delta m_k(d_k; \mu, \pi) \geq \max \{ \frac{1}{2} \| T W_k d_k \| \theta \| d_k \|^2 \} + \epsilon_1 \pi \max \{ \| c_k \|, \| r_k(d_k) \| - \| c_k \| \} \quad (2.10)
\]

holds, then terminate by returning \((d_k, \delta_k)\) and the current \( \pi. \)

4: (Termination test 2) If the residual conditions

\[
\| r_k(d_k, \delta_k) \| \leq \kappa \| c_k \| \quad \text{and} \quad \| r_k(d_k) \| \leq \kappa \| c_k \|
\]

are satisfied and the curvature condition \( \frac{1}{2} T W_k d_k \geq \theta \| d_k \|^2 \) holds, then terminate by returning \((d_k, \delta_k)\) and

\[
\pi' \leftarrow (\gamma_2^T d_k + \frac{1}{2} T W_k d_k)/((1 - \tau)(\| c_k \| - \| r_k(d_k) \|)).
\]

5: (Termination test 3) If the dual displacement and feasibility measures satisfy

\[
\| r_k(0, \delta_k) \| \leq \kappa \| c_k \|, \quad \| c_k \| \leq \kappa \| c_k \| + A_k^T \lambda_k, \quad \| c_k \| \leq \kappa \| c_k \| + A_k^T \lambda_k,
\]

then terminate by returning \((0, \delta_k)\) (i.e., reset \( d_k \leftarrow 0 \)) and the current \( \pi. \)

6: (Hessian modification) If \( \Psi_k \leq \kappa_W \) and \( \frac{1}{2} d_k^T W_k d_k < \theta \| d_k \|^2 \), then modify \( W_k \leftarrow W_k + \xi D_k \), reset \( j \leftarrow 1 \) and \((d_k, \delta_k) \leftarrow (0, 0), \) and go to step 2.

7: (Search direction update) Perform one iteration of an iterative solver on \( (2.8) \) to compute an improved (approximate) solution \((d_k, \delta_k)\). Increment \( j \leftarrow j + 1 \) and go to step 2.

The inexact dogleg approach [24] similar to that in [13]; see [10] for details. It requires the inexact solution of a system similar to \( (2.8) \) with \( W_k \) replaced by a positive definite diagonal matrix.

We now present Algorithm 3. Since \( v_k \) is computed separately from \( u_k \), we now apply an iterative linear system solver to the reformulated system

\[
\begin{bmatrix} W_k & A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} d_k \\ \delta_k \end{bmatrix} = - \begin{bmatrix} \gamma_k + A_k^T \lambda_k \\ -A_k v_k \end{bmatrix}. \quad (2.12)
\]

The relative residual \( \Psi_k \) is redefined accordingly as

\[
\Psi_k(d_k, \delta_k) := \left\| \begin{bmatrix} \rho_k(d_k, \delta_k) \\ -A_k v_k + A_k d_k \end{bmatrix} \right\| / \left\| \begin{bmatrix} \gamma_k + A_k^T \lambda_k \\ -A_k v_k \end{bmatrix} \right\|.
\]

Since \( (2.12) \) stipulates \( A_k d_k = A_k v_k \), the system is consistent for suitable \( W_k \). We choose the inputs for Algorithm 3 to be the same as those used in Algorithm 2.

The values for the new constants are chosen to be \( \epsilon_2 = 0.9 \) and \( \psi = 0.1. \)

3 Algorithm Details

3.1 Switching between search direction calculations and preconditioning

(Search direction update) Perform one iteration of an iterative solver on (2.12) to compute a diagonal matrix $D_k > 0$. Compute $v_k$ by Algorithm 3 in [10]. Initialize $j \leftarrow 1$ and $(d_k, \delta_k) \leftarrow (0, 0)$.

2: (Residual test) If $j < J$ and $\Psi_k > \kappa_{\text{des}}$, then go to step 10.
3: (Direction decomposition) Set $u_k \leftarrow d_k - v_k$.
4: (Tangential component test) If

$$\|u_k\| \leq \psi \|v_k\|$$

(2.13)

or if the inequalities

$$\frac{1}{2} u_k^T W_k u_k \geq \theta \|u_k\|^2$$
$$\langle \gamma_k + W_k v_k \rangle^T u_k + \frac{1}{2} u_k^T W_k u_k \leq 0$$

(2.14a, 2.14b)

are satisfied, then continue to step 5; otherwise, go to step 8.

5: (Dual residual test) If the dual residual condition

$$\|\rho_k(d_k, \delta_k)\| \leq \kappa_1 \min \left\{ \left\| \left[ \frac{\gamma_k + A_k^T \lambda_k}{A_k v_k} \right] \right\|, \left\| \left[ \frac{\gamma_k - 1 + A_k^T \lambda_k}{A_k - 1 v_k - 1} \right] \right\| \right\}$$

(2.15)

is satisfied, then continue to step 6; otherwise, go to step 8.

6: (Termination test 1) If the model reduction condition

$$\Delta m_k(d_k; \mu, \pi) \geq \max \left\{ \frac{1}{2} u_k^T W_k u_k, \theta \|u_k\|^2 \right\} + \epsilon_1 \pi (\|c_k\| - \|r_k(v_k)\|)$$

(2.16)

is satisfied, then terminate by returning $(d_k, \delta_k)$ and the current $\pi$.

7: (Termination test 2) If $\|c_k\| - \|r_k(d_k)\| \geq \epsilon_2 (\|c_k\| - \|r_k(v_k)\|) > 0$, then terminate by returning $(d_k, \delta_k)$ and $\pi \leftarrow \max \{\pi, \pi' + \zeta\}$ where

$$\pi' \leftarrow (\gamma_k d_k + \frac{1}{2} u_k^T W_k u_k)((1 - \tau)(\|c_k\| - \|r_k(d_k)\|))$$

8: (Termination test 3) If the dual displacement $\delta_k$ yields

$$\|\rho_k(0, \delta_k)\| \leq \kappa_3 \min \left\{ \left\| \left[ \frac{\gamma_k + A_k^T \lambda_k}{A_k v_k} \right] \right\|, \left\| \left[ \frac{\gamma_k - 1 + A_k^T \lambda_k}{A_k - 1 v_k - 1} \right] \right\| \right\}$$

and the stationarity and dual feasibility measures satisfy $\|A_k^T c_k\| \leq \epsilon_3 \|\gamma_k + A_k^T \lambda_k\|$, then terminate by returning $(0, \delta_k)$ (i.e., reset $d_k \leftarrow 0$) and the current $\pi$.

9: (Hessian modification) If $\Psi_k \leq \kappa_{\text{Hess}}$, but both (2.13) and (2.14a) do not hold, then modify $W_k \leftarrow W_k + \Delta D_k$, reset $j \leftarrow 1$ and $(d_k, \delta_k) \leftarrow (0, 0)$, and go to step 2.

10: (Search direction update) Perform one iteration of an iterative solver on (2.12) to compute an improved (approximate) solution $(d_k, \delta_k)$. Increment $j \leftarrow j + 1$ and go to step 2.

[10] However, as Algorithm 2 will produce viable search directions in most practical situations and, in contrast to Algorithm 3, it only requires the inexact solution of a single linear system, it is generally advantageous to pair Algorithm 1 with Algorithm 2 rather than with Algorithm 3. Thus, our implementation computes search directions with Algorithm 2 and only switches to Algorithm 3 when there is evidence that Algorithm 2 may be unable to produce a productive search direction.

Our trigger for switching between the two search direction algorithms is based on the steplength obtained as a result of the line search. If during a given iteration of Algorithm 1, Algorithm 2 has been employed for computing the search direction and the line search produces a steplength below a given threshold $\alpha_1$, then this may be an indication that $A_k$ is losing rank, causing the steps to become too large. (Of course, the short steplength may simply be due to the nonlinearities of...
the problem functions themselves, but even in that case the algorithm may benefit by employing Algorithm 3. In such cases, we decide to employ Algorithm 3 in the following iteration of Algorithm 1 and continue to employ it until an iteration yields a steplength above $\bar{\alpha}_1$. The motivation for this scheme is that Algorithm 1 paired with Algorithm 2 is guaranteed to converge for an equality-constrained problem (e.g., a given barrier subproblem) under common assumptions, as long as the constraint Jacobians have full row rank and their smallest singular values are bounded away from zero. Specifically, for the analysis of Algorithm 2 in [9], the latter requirement is used to show that the steplength $\alpha_k$ is bounded away from zero. Thus, we use a small steplength $\alpha_k$ as an indicator to switch to Algorithm 3.

In our implementation, we choose the threshold value to be $\bar{\alpha}_1 = 10^{-3}$.

When an iterative solver is used to solve the linear systems (2.8) and (2.12), an effective preconditioner is essential to keep the number of iterations low. For best performance, the preconditioner should be tailored to specific problems. However, for our numerical experiments, we use a general-purpose preconditioning approach. The details of our preconditioning method can be found in [10,25]. In short, the approach is based on an algebraic multi-level preconditioner designed for symmetric highly indefinite systems. It uses symmetric maximum weight matchings to improve the block diagonal dominance of the system, followed by an inverse-based pivoting strategy to compute an inexact factorization. In order to bound the norm of the inverse, the factorization of some rows and columns might be postponed to the end. This leaves a Schur complement to which the procedure is applied recursively within a multilevel preconditioning framework.

For the iterative solution of linear systems we use the symmetric quasi-minimum residual (SQMR) method [16] which has been found to work well with this preconditioner. Here, we allow a depth up to 30 in the multi-level approach [25], the constant bounding the norm of the inverse of the factor is chosen to be $\kappa_L = 2$, and the drop tolerances for the factor and the Schur complement are set to be $\epsilon_L = 10^{-2}$ and $\epsilon_S = 10^{-3}$, respectively. The SQMR method is allowed a maximum number of 1500 iterations. If this number is exceeded, then the preconditioner is recomputed with tightened drop tolerances (both divided by 3) and the iteration counter is reset. If necessary, the tolerances are tightened repeatedly. If an acceptable solution for Algorithm 2 has not been computed after 4 such attempts, then the method reverts to Algorithm 3. If an acceptable solution for Algorithm 3 has not been computed after 4 such attempts, then the last computed inexact solution is used (without guarantees for a successful line search). In either of these latter two cases, before a new linear system is solved, the drop tolerances are multiplied by 3, though they are never set higher than the default values given above.

3.2 Hessian modification strategy and flexible penalty function

In the definition of the Hessian matrix $W_k$ in (2.7), the choice $\Xi_k = \mu S_k^{-2}$ corresponds to the so-called primal interior-point iteration. This was considered for the global convergence analysis in [13]. However, our implementation follows the more efficient primal-dual interior-point strategy $\Xi_k = S_k^{-1} Y_k$, where $Y_k = \text{diag}(y_k)$ with dual variables $y_k$ corresponding to the slack bounds (i.e. $s \geq 0$). It is easy to see that the analysis in [13] still holds as long as $\nu \mu S_k^{-2} \succeq \Xi_k \succeq \nu \mu S_k^{-2}$ for some
constants $\nu \geq 1 \geq \nu > 0$; we choose $\nu = 10^{10}$ and $\nu = 10^{-10}$ in our experiments. This is achieved by adjusting $y_k$, if necessary, after each iteration; see [28].

Our strategy for modifying the Hessian in Algorithms 2 and 3 is analogous to the one described in [28], where a multiple of the identity is added to the unscaled Hessian matrix. This corresponds to using $D_k$ with a $(1,1)$-block being $I$ and a $(2,2)$-block being $\Sigma^2_k$. Furthermore, we choose $\xi$ according to the strategy for choosing $\delta w$ in Algorithm IC in [28]. However, in contrast to Algorithm IC, our trigger for a modification is not the inertia of the primal-dual system. Rather, we trigger a modification based on the conditions described in step 6 of Algorithm 2 and step 9 of Algorithm 3. We have also found it beneficial to trigger a modification at the start of the search direction computation if in the previous iteration the line search reduced $\alpha_k$ due to the sufficient decrease condition (2.5). This leads to somewhat shorter search directions and makes the acceptance of larger steplengths more likely, often leading to a reduction in iteration count.

An important algorithmic feature of our code is the use of a flexible penalty function [11]. This mechanism is designed to avoid a pitfall of penalty functions, namely the potential for the algorithm to set an unnecessarily large value of the penalty parameter and thus restrict the iterates to remain close to the feasible region. This can lead to small steplengths and slow convergence.

The effect of the flexible penalty function on our line search is that, instead of requiring $\alpha_k$ to satisfy the sufficient decrease condition (2.5) for a fixed $\pi$, we only require that $\alpha_k$ satisfies a sufficient decrease condition for some $\pi$ in an interval $[\pi^l, \pi^u]$. In particular, given $\pi^l \leq \pi^m \leq \pi^u$, $\alpha_k \in (0, \alpha_k^{\text{max}}]$ is acceptable as long as

$$
\phi(z_k + \alpha_k d_k; \mu, \pi) \leq \phi(z_k; \mu, \pi) - \eta_2 \Delta m_k(d_k; \mu, \pi^m) \quad \text{for some} \quad \pi \in [\pi^l, \pi^u].
$$

(3.1)

We have adapted the strategy proposed in [11] for updating $\pi^l$ and $\pi^u$ and for setting $\pi^m$ during each iteration. In particular, our updates for $\pi^l$ and $\pi^u$ are essentially those in [11]; the details can be found in [10]. As for the choice of $\pi^m$, we consider two cases. If in the current iteration Termination test 2 was satisfied, but Termination test 1 was not, then we follow [11] and set $\pi^m$ to be the maximum of $\pi^l$ and $\pi^t$, where $\pi^t$ is computed during Termination test 2. This choice guarantees that the model reduction $\Delta m_k(d_k; \mu, \pi^m)$ is positive. Otherwise, if Termination test 1 was satisfied, then we set $\pi^m$ to be $\pi^l$ since, based on our decision to use $\pi^l$ as the penalty parameter value in (2.10) and (2.16), this choice also guarantees that the model reduction $\Delta m_k(d_k; \mu, \pi^m)$ is positive. Overall, we guarantee that (3.1) is a sufficient decrease condition for $\phi$ in either case.

4 Numerical Experiments

The algorithm described in the previous sections was implemented in the IPOPT open-source package (http://www.coin-or.org/Ipopt/); for our experiments we use revision 1954 of the branches/parallel development branch. The linear systems are solved using the iterative linear system solvers and preconditioners implemented in PARDISO (http://www.pardiso-project.org/) version 4.1.1. The finite-element discretization of the PDEs in §4.1-4.2 was implemented using the open-source libmesh library [22], revision 3881 in its trunk branch, together with the PETSc library [2] version 3.1-p3. The three-dimensional meshes for the example in §4.2 are generated with the tetgen software (http://tetgen.berlios.de/).
In IPOPT, we use default parameter settings with a termination tolerance $10^{-6}$, together with the parameter choices given in the previous sections. The iterative linear solver in PARDISO uses SQMR \[16\] with the preconditioner described in \S3.1.

To assess the robustness of the algorithm we compare its performance with the default method in IPOPT on AMPL \[15\] versions of problems from the CUTE set \[17\]; details of the experiment can be found in \[10\]. Algorithm 1 is able to solve 89% of the 617 problems, and is therefore almost as robust as the default algorithm in IPOPT that solves 96%. Note that for 142 problems, the algorithm switches to Algorithm 3 at some point, indicating that the switching strategy in \S3.1 was utilized by the algorithm. In contrast, the method proposed in \[13\], which always uses Algorithm 3, has a success rate of only 84%, meaning that we have achieved an increase in robustness of 5% by utilizing Algorithm 2.

The numerical experiments in the rest of this section illustrate the performance of our implementation on two PDE-constrained problems. We show that our method provides improved computation times compared to the default IPOPT algorithm. The results were obtained on 8-core Intel Xeon machines with 2.33GHz clock speed and 32GB RAM, running Ubuntu Linux with GNU 4.4.1 compilers.

### 4.1 Optimal boundary control

Our first PDE-constrained optimization problem is an optimal control problem motivated by the “Heating with radiation boundary conditions” example in Section 1.3.1 of \[27\]. The PDE is to satisfy $-\Delta T = 0$, where $T$ denotes temperature, in a domain $\Omega \subseteq \mathbb{R}^3$. A suitable boundary condition is to satisfy \[\frac{\partial T}{\partial n} = \chi (u - T^4)\] on the boundary $\Gamma$ of $\Omega$. This condition expresses the radiation heat loss according to the Stefan-Boltzmann law with a Stefan’s constant $\chi > 0$, where the control $u \geq 0$ dictates heat that can be resupplied on $\Gamma$. The goal is to minimize the amount of heat supplied, i.e., $\int_{\Gamma} u \, da$, while attaining a temperature of $T_{min,j}$ or more within $N_{S}$ subregions $\Omega_j \subseteq \Omega$. We multiply $-\Delta T = 0$ with a test function $v \in H^1(\Omega)$ and apply Green’s formula together with the boundary condition. The weak formulation of the PDE is then to find $T \in H^1(\Omega)$ such that

$$0 = -\int_{\Omega} \Delta T v \, dx = \int_{\Omega} \nabla T \cdot \nabla v \, dx - \chi \int_{\Gamma} (T^4 - u) v \, da \quad \forall v \in H^1(\Omega). \quad (4.1)$$

We generate a regular mesh of tetrahedrons, each with volume $h^3/24$ for $h > 0$, and use the standard linear finite element basis functions \{\varphi_i\}_{i=1,\ldots,n_h}$. Projecting (4.1) onto the generated finite dimensional subspace $V^h$ by approximating $T$ with $T^h = \sum_i T^{(i)} \varphi_i$ and $u$ by $u^h = \sum_i u^{(i)} \varphi_i$ (the latter requires discretized values $u^{(i)}$ only corresponding to the boundary $\Gamma$), we solve the finite-dimensional problem

$$\min_{u^{(i),T^{(i)}}} \sum_i u^{(i)} \int_{\Gamma} \varphi_i \, da$$

subject to $u^{(i)} \geq 0$ and

$$0 = \int_{\Omega} \sum_i T^{(i)} \nabla \varphi_i \cdot \nabla \varphi_j \, dx - \chi \int_{\Gamma} \left(\sum_i T^{(i)} \varphi_i\right)^4 - \sum_i u^{(i)} \varphi_i \right) \varphi_j \, da \quad \forall j \quad (4.2a)$$

$$T^{(i)} \geq T^{(i)}_{min} \quad \text{for } j \in \{1,\ldots,N_S\} \text{ and } i \in \{i \mid \exists x \in \Omega_j : \varphi_i(x) = 1\}. \quad (4.2b)$$
We choose $\chi = 1$ and $\Omega = (0,1)^3$ and define two regions to be heated, $\Omega_1 = [0.1, 0.2] \times [0.05, 0.3] \times [0, 0.1]$ and $\Omega_2 = [0.8, 1] \times [0.75, 1] \times [0.7, 1]$, with associated threshold temperatures of $T_1^{\min} = 2.5$ and $T_2^{\min} = 2$. In (4.2b), we used the fact that a nodal finite element basis was chosen, so that $\max_{x \in \Omega} \phi_i(x) = 1$, and for all $x \in \Omega$ we have $\sum_i \phi_i(x) = 1$. Since $\nabla \phi_i \cdot \nabla \phi_j = \mathcal{O}(1/h^2)$ and $\int_E dx = \mathcal{O}(h^3)$ for a tetrahedron $E$, we multiply (4.2a) by $10^{-2}/h$ in our implementation to ensure that the gradients of these constraints do not vanish as $h \to 0$. Similarly, the objective function was scaled internally by the factor $10^{-2}/(h^2)$.

We executed the optimization algorithm for three choices of the discretization level. As initial point, we chose $T = T_{\text{init}}$ with $T_{\text{init}} = 1.1(T_1^{\min} + T_2^{\min})$, and $u = (T_{\text{init}})^4$. Table 4.1 shows the discretization parameter ($h$), number of optimization variables (#var), number of simple bound constraints (#bds), number of equality constraints (#eq), and number of inequality constraints (#ineq) for various instances of this example. Table 4.2 provides performance measures in the form of number of iterations (it), final objective value $f(x_*)$, CPU seconds (CPUs), and CPU seconds per iteration (CPUs/it) for the default IPOPT algorithm and for Algorithm 4. The last column shows the overall CPU time speedup obtained by the inexact algorithm. We see a significant gain in computation speed that becomes more pronounced as the problem size increases. For the largest problem, the speedup is a factor of 7.85. The corresponding solution is depicted in Figure 4.1.

Table 4.2 lists the average CPU time per iteration, but it should be noted that the step computation requires considerably more time towards the end of the optimization procedure. Considering the case $h = 0.02$, in the first 22 IPOPT iterations the preconditioners (each computed in less than one minute) have fill-in factors of at most 3 and SQMR requires only between 35 and 200 iterations, leading...
On the Implementation of an Inexact Interior-Point Algorithm

### Table 4.1 Problem sizes for instances of the example in §4.1.

<table>
<thead>
<tr>
<th>$h$</th>
<th>$#\text{var}$</th>
<th>$#\text{bds}$</th>
<th>$#\text{eq}$</th>
<th>$#\text{ineq}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.04</td>
<td>89453</td>
<td>9183</td>
<td>81951</td>
<td>0</td>
</tr>
<tr>
<td>0.03</td>
<td>199389</td>
<td>16498</td>
<td>186319</td>
<td>0</td>
</tr>
<tr>
<td>0.02</td>
<td>670153</td>
<td>42313</td>
<td>640151</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 4.2 Performance measures for the example in §4.1.

<table>
<thead>
<tr>
<th>$h$</th>
<th>alg</th>
<th>it</th>
<th>$f(x^*)$</th>
<th>CPUs</th>
<th>CPUs/it</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.04</td>
<td>IPOPT</td>
<td>34</td>
<td>39.9458</td>
<td>2806.16</td>
<td>85.04</td>
<td></td>
</tr>
<tr>
<td>0.04</td>
<td>Alg.</td>
<td>33</td>
<td>39.9458</td>
<td>646.77</td>
<td>19.60</td>
<td>4.34</td>
</tr>
<tr>
<td>0.03</td>
<td>IPOPT</td>
<td>34</td>
<td>37.7909</td>
<td>16330.56</td>
<td>480.31</td>
<td></td>
</tr>
<tr>
<td>0.03</td>
<td>Alg.</td>
<td>37</td>
<td>37.7909</td>
<td>4495.83</td>
<td>121.51</td>
<td>3.63</td>
</tr>
<tr>
<td>0.02</td>
<td>IPOPT</td>
<td>46</td>
<td>40.9115</td>
<td>304780.45</td>
<td>6625.66</td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td>Alg.</td>
<td>47</td>
<td>40.9115</td>
<td>38824.33</td>
<td>826.05</td>
<td>7.85</td>
</tr>
</tbody>
</table>

To less than 4 minutes for each step computation. However, in the final IPOPT iterations, the dropping tolerances have to be tightened. Then, the preconditioner (computed in up to 9 minutes) has a fill-in factor of almost 10 and SQMR requires more than 1000 iterations, leading to times up to 35 minutes per step computation.

#### 4.2 Server room cooling

Our second example is motivated by the problem of cooling computer equipment. In our simplified model, we assume that cold air is blown into a room from air conditioners (ACs), and that air leaves the room at exhausts. Inside the domain lies equipment with hot surfaces that need to be cooled by air passing alongside.

For simplicity, we suppose that air is incompressible, has no internal friction, and that all velocities are far below the speed of sound. Under these assumptions, we can model air velocity $y(x)$ as the gradient of a potential $\Phi(x)$ satisfying the Laplace equation $-\Delta \Phi = 0$ in $\Omega \subseteq \mathbb{R}^3$. Appropriate boundary conditions for the walls (including non-heat producing surfaces of the equipment) $\Gamma_W$ and for the heat producing surfaces $\Gamma_T$ of the equipment are $\frac{\partial \Phi}{\partial n} = 0$ on $\Gamma_W \cup \Gamma_T$. Similarly, boundary conditions for the cold air inlets are $\frac{\partial \Phi}{\partial n} = -u_{AC_i} \Psi_{\Gamma_{AC_i}}$ on $\Gamma_{AC_i}$, and for the exhausts are $\frac{\partial \Phi}{\partial n} = u_{Ex_i} \Psi_{\Gamma_{Ex_i}}$ on $\Gamma_{Ex_i}$. Here, $\Psi_{\Gamma_{AC_i}}(x)$ ($i = 1, \ldots, N_{AC}$) and $\Psi_{\Gamma_{Ex_i}}(x)$ ($i = 1, \ldots, N_{Ex}$) define velocity profiles on the surfaces of the ACs and exhausts, respectively. Similarly, $u_{AC_i} \in \mathbb{R}$ and $u_{Ex_i} \in \mathbb{R}$ denote control parameters for the maximal flow rates at these air inlets and outlets. The weak formulation of the PDE is to find $\Phi \in H^1(\Omega)$ such that, for all $v \in H^1(\Omega)$, we have

$$0 = \int_\Omega \nabla \Phi \cdot \nabla v \, dx + \sum_{i=1}^{N_{AC}} \int_{\Gamma_{AC_i}} u_{AC_i} \Psi_{\Gamma_{AC_i}} v \, da - \sum_{i=1}^{N_{Ex}} \int_{\Gamma_{Ex_i}} u_{Ex_i} \Psi_{\Gamma_{Ex_i}} v \, da. \quad (4.3)$$

It is important to note that (4.3) has a solution only if the controls satisfy the mass balance equation

$$\sum_{i=1}^{N_{AC}} \int_{\Gamma_{AC_i}} u_{AC_i} \Psi_{\Gamma_{AC_i}} \, da - \sum_{i=1}^{N_{Ex}} \int_{\Gamma_{Ex_i}} u_{Ex_i} \Psi_{\Gamma_{Ex_i}} \, da = 0, \quad (4.4)$$
and in that case (4.3) only determines the potential $\Phi \in H^1(\Omega)$ up to an additive constant. Therefore, a normalization condition will be introduced below.

As a constraint, we require that the air speed at the heat-producing surfaces has a minimum velocity so that heat is carried away. More precisely, recalling that the velocity is the gradient of the potential function $\Phi$, we impose the point-wise state constraint $\|\nabla \Phi(x)\|^2 \geq y_{\min}^2$ for all $x \in \Gamma_T$ with a constant $y_{\min} > 0$.

To obtain the discretized problem, we generate an irregular mesh of tetrahedrons, each with maximal volume $h^3$, again choose a finite-dimensional subset $V^h \subseteq H^1(\Omega)$ with a basis $\{\phi_i\}_{i=1,\ldots,n_h}$, and express the finite-dimensional approximation $\Phi_h$ of $\Phi = \sum_i \phi_i^T \phi_i$ with coefficients $\phi_i \in \mathbb{R}^{n_h}$. Defining $u = (u_{AC}^i, u_{Ex})$ as the vector consisting of all control parameters, the discretized PDE (4.3) then becomes $A\phi - Bu + \gamma e \bar{u} = 0$, where $A$ denotes the stiffness matrix $A^{i,j} = \int_{\Omega} \nabla \phi_i \cdot \nabla \phi_j dx$, and $B = [BA \quad BE_{Ex}]$ implements the boundary conditions with $B_{AC}^{i,j} = -\int_{\Gamma_{AC}} \Psi_{AC} \phi_i \phi_j da$ and $B_{Ex}^{i,j} = \int_{\Gamma_{Ex}} \Psi_{Ex} \phi_i da$.

The finite-dimensional optimization problem is

$$\min_{\phi, u, \bar{u}} \sum \beta_j u_{AC}^j$$

subject to $u \geq 0$ and

$$A\phi - Bu + \gamma e \bar{u} = 0 \quad (4.5a)$$
$$\gamma e^T \phi - \bar{u} = 0 \quad (4.5b)$$
$$e^T Bu = 0 \quad (4.5c)$$
$$\int_{\Gamma_e} \nabla \phi(x) \cdot \nabla \phi(x) da - y_{\min}^2 \left( \int_{\Gamma_e} da \right) \geq 0 \text{ for } \Gamma_e \subseteq \Gamma_T \quad (4.5d)$$

with weights $\beta_j > 0$ in the objective function, and $e = (1, \ldots, 1)^T \in \mathbb{R}^{n_h}$. Here, $\{1, 2, \ldots, 6\}$ is a compact way of writing (4.4), and (4.5a) is the discretized version of the point-wise state constraint, which is posed for all element faces $\Gamma_e$ contained in a heat producing surface $\Gamma_T$. Note that the constraint (4.5d) is nonlinear and nonconvex. Again, in our implementation of the above problem, we scaled the constraints (4.5a) and (4.5b) by factors $10^{-2}/h$ and $10^{-1}/h$, respectively, to ensure that the gradients of these functions do not vanish as $h \to 0$.

To overcome the ill-posedness of the PDE, an auxiliary variable $\bar{u} \in \mathbb{R}$ has been added. Equation (4.5a) includes the discretized PDE, where $\gamma e \bar{u}$ acts as a virtual source or sink over $\Omega$. Since we impose mass conservation in (4.5c), this term yields $\bar{u} = 0$. Furthermore, an integral-type equation is imposed in (4.5b). Indeed, $e^T \phi$ is a discretization of $\int_{\Omega} \phi d\mu$ for some measure $\mu$ depending on the finite-element discretization and is set to zero in (4.5b) since $\bar{u} = 0$, therefore normalizing $\Phi$.

For our experiments, we choose $\beta_j = 1$, $\gamma = 1$, $\bar{u} = 10^8$, $y_{\min} = 1$, $\Gamma_{AC} = \{0\} \times [0.4, 0.6] \times [0.2, 0.4]$, $\Gamma_{AC} = [0.4, 0.6] \times \{0\} \times [0.2, 0.4]$, $\Gamma_{AC} = 1 \times [0.4, 0.6] \times [0.6, 0.8]$. The equipment is placed so that $\Omega_{E_{\bar{u}}} = [0.2, 0.7] \times [0.2, 0.4] \times [0.8]$ and $\Omega_{E_{\bar{u}}} = [0.2, 0.6] \times [0.6, 0.8] \times [0.8]$ with the remaining boundaries $\Gamma_F$ and $\Gamma_W$ defined as illustrated in Figure 4.2.

The airflow at the inlets and outlets are defined to have quadratic profiles.

Due to the nooks in $\Omega$ created by the equipment, numerical experiments with linear finite elements showed only linear $L^2$-convergence of the PDE solution as $h \to 0$. However, since the point-wise state constraint involves the gradient of the conditions.

The airflows at the inlets and outlets are defined to have quadratic profiles.
Fig. 4.2 Optimal solution of the server room cooling optimization example. On the left, we see the streamlines of the airflow, going from the main AC on the left to the exhaust on the right. On the right, we have a bottom view of the domain $\Omega$, where the colors have been chosen to be dark if the air velocity is close to the threshold $y_{\text{min}} = 1$. One can clearly see a region at the wall of the larger piece of equipment, at which the velocity is close to critical, indicating the location of the active constraints (4.5d) in $\Gamma_T$.

Table 4.3 Problem sizes for instances of the example in §4.2.

<table>
<thead>
<tr>
<th>$h$</th>
<th>#var</th>
<th>#bds</th>
<th>#eq</th>
<th>#ineq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.03</td>
<td>88398</td>
<td>4</td>
<td>88395</td>
<td>1528</td>
</tr>
<tr>
<td>0.02</td>
<td>285510</td>
<td>4</td>
<td>285507</td>
<td>3409</td>
</tr>
<tr>
<td>0.015</td>
<td>663886</td>
<td>4</td>
<td>663883</td>
<td>6110</td>
</tr>
</tbody>
</table>

Table 4.4 Performance measures for the example in §4.2.

<table>
<thead>
<tr>
<th>$h$</th>
<th>alg</th>
<th>it</th>
<th>$f(x_\ast)$</th>
<th>CPUs</th>
<th>CPUs/it</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.03</td>
<td>IPOPT</td>
<td>21</td>
<td>15.5283</td>
<td>4511.48</td>
<td>214.83</td>
<td></td>
</tr>
<tr>
<td>0.03</td>
<td>Alg[1]</td>
<td>24</td>
<td>15.5283</td>
<td>1710.30</td>
<td>71.26</td>
<td>2.64</td>
</tr>
<tr>
<td>0.02</td>
<td>IPOPT</td>
<td>33</td>
<td>15.5694</td>
<td>69427.33</td>
<td>2103.86</td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td>Alg[1]</td>
<td>28</td>
<td>15.5694</td>
<td>10008.50</td>
<td>357.45</td>
<td>6.94</td>
</tr>
<tr>
<td>0.015</td>
<td>IPOPT</td>
<td>32</td>
<td>15.6509</td>
<td>328320.22</td>
<td>16510.01</td>
<td></td>
</tr>
<tr>
<td>0.015</td>
<td>Alg[1]</td>
<td>27</td>
<td>15.6509</td>
<td>29526.53</td>
<td>1093.58</td>
<td>17.89</td>
</tr>
</tbody>
</table>

In this example, the default settings for the preconditioner thresholds were sufficient in each iteration, so no tightening occurred. For the $h = 0.015$ case, the computation time for the preconditioner ranged from 164 to 234 seconds (with an average of 204 seconds), the number of SQMR iterations was 204–1129 with an
average of 396, and the time spent in SQMR ranged from 365 to 2018 seconds (with an average of 719 seconds). While there was some variation, we did not observe a degeneration of computation time per step computation as in §4.1.

5 Conclusion and Final Remarks

We have presented a detailed description of an implementation of a primal-dual interior-point method for large-scale nonconvex optimization where the search directions are computed inexactly by means of an iterative linear system solver. Ideally, the algorithm computes a search direction through the inexact solution of a single linear system (as in [9]). However, when appropriate, it falls back on the step decomposition strategy proposed in [13] so that, overall, the strong global convergence properties presented in [13] are attained. Numerical experiments on a large set of test problems and on two PDE-constrained optimization problems have also been presented. These results demonstrate the robustness of the approach and illustrate the significant speedup our algorithm attains when compared to an algorithm based on direct factorizations of the primal-dual system matrices.

As mentioned at the end of §4.1, we have observed a decrease in effectiveness of the preconditioner as the barrier parameter approaches zero. It remains a subject of future research to explore whether the specific structure of the ill-conditioning caused by the log-barrier terms can be handled efficiently within the incomplete-factorization-based preconditioner described in §3.1.

Acknowledgments

The authors thank Michael Henderson, Vanessa Lopez, and Ulisses Mello for suggesting the server room cooling example, and Madam Sathe for discussions regarding the preconditioner. They also thank the referees whose suggestions led to significant improvements. Johannes Huber gratefully acknowledges financial support from IBM during his summer internship at the IBM Watson Research Center.

References


