NORTHWESTERN UNIVERSITY

Practical Techniques for Nonlinear Optimization

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Sciences

By

Long Hei

EVANSTON, ILLINOIS

June 2007

ABSTRACT

Practical Techniques for Nonlinear Optimization

Long Hei

At the heart of nonlinear optimization methods lies the solution of linear systems of equations. As the size of the problem increases, it is imperative to use iterative methods, such as the conjugate gradient algorithm, to solve these linear systems. In the context of constrained optimization, it has proved to be effective to use the projected conjugate gradient method to solve the equality constrained quadratic subproblems used to generate a step. In the first part of this thesis we present new preconditioners for the projected conjugate gradient method used in interior point methods. They fall under the category of "constraint preconditioners" and make use of the so-called condensed system to exploit the structure arising in interior point methods. We also give attention to the requirements imposed by trust region techniques.

In the second part of the thesis we study nonlinear programming formulations and numerical methods for solving a strategic bidding problem in a short term electricity market. This problem can be formulated as a bilevel program whose lower level problem is linear. We analyze the properties of the constraints of the strong duality formulation of this bilevel program and show that the LICQ and MFCQ constraint qualification conditions fail at every feasible point. Therefore, even finding local minimizers is a difficult problem. We then consider the use of nonlinear programming algorithms for solving the bilevel program. They are appealing because they are well suited for large problems, but since the objective function of the strategic bidding problem is discontinuous and has many local minima, we consider

heuristics (based on multistart) to attempt to find a solution. We report good solutions in acceptable time.

The thesis concludes with some observations about the practical limitations of filter techniques.

ACKNOWLEDGMENTS

I am extremely grateful to have the privilege of working with my advisor Jorge Nocedal. Jorge is an intelligent and energetic advisor, a knowledgeable and enthusiastic teacher, and also a caring and encouraging mentor. I thank him for his generous spiritual and financial support, for his immense patience and tolerance throughout my years at Northwestern University, especially during the most difficult period of time.

My appreciation is also expressed to the other members of my graduate committee, Robert Fourer, Sanjay Mehrotra and Richard Waltz, for their valuable comments on this thesis. I would also like to thank Richard for his input in the implementation of the preconditioning ideas, as well as for being a great mentor in the early stages of my graduate study.

I thank Professor Ya-xiang Yuan for introducing me to the area of nonlinear programming. I also express my gratitude to many people in the optimization community, whom I have had the pleasure of learning from or collaborating with, including but not limited to: Richard Byrd, Sven Leyffer, Steve Benson, Jose Luis Morales, Dominique Orban and Guanghui Liu.

The people in the IEMS Department deserve special acknowledgment for providing an enjoyable and friendly working atmosphere and for their support in various ways during these years. Instead of listing many names, I single out Gabriel López-Calva and Frank Curtis – they are great officemates and colleagues. Also, I would like to thank many friends from outside the department, outside the university and even outside the country.

This work is dedicated to my parents and my brother for their never-ending love and constant support to me, and to my wife Wenli Lv for her immeasurable help in many aspects of my life over the past years, especially in finding more than what is contained in research work. This thesis would not exist without her effort and support.

This work was supported in part by the National Science Foundation Grant CCF-0514772 and by Department of Energy Grant DE-FG02-87ER25047-A004.

Contents

List of Tables

List of Figures

Chapter 1

Background on Nonlinear Optimization

1.1 CG Method for Unconstrained Optimization

We consider the linear system of equations

$$Ax = b, \tag{1.1}$$

where $A \in \Re^{n \times n}$ is invertible and $x, b \in \Re^n$. Let $x^* = A^{-1}b$ be the exact solution of (1.1).

There are two classes of methods for solving (1.1): direct methods and iterative methods. The former performs certain types of factorization to the coefficient matrix A, and then backsolves the variables x directly. The latter does not rely on any factorization of A. Instead, it generates a series of iterates $\{x_1, x_2, \cdots, x_k, \cdots\}$ that approaches x^*. See Golub and Van Loan [20] and references therein. Normally iterative methods only require us to compute certain matrix-vector products. For example, Krylov space methods, an important class of iterative methods, only require the computation of Av for any vector $v \in \Re^n$ in each

iteration.

When the number of variables n increases, the factorization performed to A may become prohibitively expensive. Therefore iterative methods have received much attention over the last decades.

1.1.1 The CG Algorithm

The Conjugate Gradient (CG) algorithm is a special Krylov space method that solves symmetric and positive definite linear system of equations. If $A \in \Re^{n\times n}$ is symmetric and positive definite, linear systems of equations (1.1) is equivalent to an unconstrained Quadratic Program (QP) of the form

$$\underset{x\in\Re^n}{\text{minimize}} \quad q(x) = \tfrac{1}{2}x^T Ax - b^T x. \tag{1.2}$$

The CG algorithm is shown in Algorithm 1.1.

Algorithm 1.1 *(CG algorithm for* (1.1)*) Choose* x_0*, compute* $r_0 = Ax_0 - b$*, define* $p_0 = -r_0$*,* $k = 0$*. Repeat until convergence*

$$\alpha_k = (r_k^T r_k)/(p_k^T A p_k) \tag{1.3a}$$

$$x_{k+1} = x_k + \alpha_k p_k \tag{1.3b}$$

$$r_{k+1} = r_k + \alpha_k A p_k \tag{1.3c}$$

$$\beta_{k+1} = (r_{k+1}^T r_{k+1})/(r_k^T r_k) \tag{1.3d}$$

$$p_{k+1} = -r_{k+1} + \beta_{k+1} p_k \tag{1.3e}$$

$$k = k + 1. \tag{1.3f}$$

In this algorithm $\{p_k\}$ is a series of conjugate directions of A. In each iteration an exact line search is performed along p_k and the steplength is α_k.

We now list a few properties of the CG algorithm. Detailed discussion on Algorithm 1.1 can be found in Nocedal and Wright [33].

1. The quadratic function value $q(x_k)$ at the iterates decreases monotonically. That is, $q(x_{k+1}) \leq q(x_k)$ for any $k \geq 0$.

2. The norm of the iterates increases monotonically if $x_0 = 0$. That is, $\|x_{k+1}\| \geq \|x_k\|$ for any $k \geq 0$.

3. The convergence rate of the CG algorithm heavily depends on the eigenvalue distribution of matrix A. If matrix A has r distinct eigenvalues, in exact arithmetic Algorithm 1.1 terminates at the solution of (1.1) or (1.2) in at most r iterations. Furthermore, the CG algorithm converges fast when matrix A has clustered eigenvalues.

We emphasize that Algorithm 1.1 is designed for the case when A is positive definite, which corresponds to a convex quadratic program of the form (1.2). When A is symmetric but has negative eigenvalues, problem (1.2) does not have a finite solution in $\Re^n$. This can be easily shown by the fact that the value of $q(x)$ decreases along the negative eigendirection of A.

1.1.2 CG with a Trust Region

Now let us consider a quadratic program with a trust region constraint

$$\underset{x \in \Re^n}{\text{minimize}} \quad q(x) = \tfrac{1}{2} x^T A x - b^T x \tag{1.4a}$$

$$\text{subject to} \quad \|Bx\| \leq \Delta, \tag{1.4b}$$

where $A \in \Re^{n \times n}$ is symmetric and $B \in \Re^{n \times n}$ is nonsingular. When B is the identity matrix or a multiple of it, constraint (1.4b) is a circular trust region; otherwise it is an elliptical trust region.

We can try to solve problem (1.4) using an iterative method such as CG. Because of the trust region constraint (1.4b), the solution of problem (1.4) does not necessarily satisfy $Ax = b$. Hence our goal is to find the (local) minimum of function $q(x)$ in the trust region. Note that a local minimum of $q(x)$ in the trust region exists even if A has negative eigenvalues. In other words, the quadratic problem (1.4) has a local solution even if it is non-convex. The usual starting point is $x_0 = 0$.

When the trust region is circular (for example, $B = I$), we can take the advantage of the properties of Algorithm 1.1 and solve problem (1.4) efficiently. We know that the quadratic model value $q(x)$ decreases monotonically while $\|x\|$ increases monotonically along all the CG iterates, hence whenever we find a certain CG iterate x_k such that $\|x_k\| \geq \Delta$ we know that no further CG iterate will be feasible to (1.4b), and that no previous CG iterate gives lower quadratic model values than $q(x_k)$. For these reasons it is safe to terminate the CG iterations.

Based on these arguments, Steihaug [40] suggests to solve (1.4) using a modified version of Algorithm 1.1 when the trust region is circular. There are two modification rules to the CG algorithm:

1. If the search direction is a negative curvature direction, defined as a direction p such that $p^T Ap < 0$, the steplength should be chosen such that the next iterate lies on the trust region bound.

2. If the exact line search is such that the new iterate lies outside the trust region, the steplength should be chosen such that the next iterate lies on the trust region bound.

Unfortunately, the above rules do not work well for problems of the form (1.4) with

elliptical trust regions (i.e. $B \neq \gamma I$ for any γ). The reason is: If we still apply CG to problem (1.4), the generated iterates x_k will still give decreasing quadratic model values $q(x_k)$, but we only know that $\|x_k\|$ increases monotonically, while $\|Bx_k\|$ does not necessarily increase monotonically. Hence, even if we find a CG iterate x_k such that $\|Bx_k\| \geq \Delta$, there could be a further CG iterate that is feasible to (1.4b) while giving even lower quadratic model values. If we terminate the CG iterations at x_k, as suggested by the regular rules above, it is possible that it is only a poor solution estimate to the problem and we could find a better solution if we keep running CG.

One possible way to solve (1.4) with elliptical trust regions is the following: Since B is nonsingular, we introduce change of variables $\hat{x} = Bx$ or $x = B^{-1}\hat{x}$, and thus in terms of the new variables $\hat{x}$, problem (1.4) becomes

$$\underset{\hat{x} \in \Re^n}{\text{minimize}} \quad \hat{q}(\hat{x}) = \tfrac{1}{2}\hat{x}^T(B^{-T}AB^{-1})\hat{x} - (B^{-T}b)^T\hat{x} \tag{1.5a}$$

$$\text{subject to} \quad \|\hat{x}\| \leq \Delta. \tag{1.5b}$$

Problem (1.5) has a circular trust region, which can be solved efficiently by the CG algorithm with Steihaug's modification rules. The solution x of problem (1.4) is then computed by $x = B^{-1}\hat{x}$ where $\hat{x}$ is the solution of (1.5).

We note that this approach works for (1.4) with circular trust regions as well, where the change of variables $\hat{x} = Bx$ amounts to "no change of variables" when $B = I$.

1.2 Equality Constrained Quadratic Programs

The following problem is an Equality constrained Quadratic Program (EQP):

$$\underset{x\in\Re^n}{\text{minimize}} \quad \tfrac{1}{2}x^T Hx + c^T x \tag{1.6a}$$

$$\text{subject to} \quad Ax = b, \tag{1.6b}$$

where $H \in \Re^{n\times n}$ is symmetric, $A \in \Re^{m\times n}$, $c \in \Re^n$ and $b \in \Re^m$. We assume that H is positive definite in the null space of A. Note that the first order necessary conditions of (1.6) form a linear system of equations of the form

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix}. \tag{1.7}$$

As is well known, the coefficient matrix in (1.7) is indefinite. Equation (1.7) is sometimes referred to as a *saddle point problem* or a *KKT system*.

There are three classes of methods to solve problem (1.6):

- Direct method: We can factorize the coefficient matrix in (1.7) and solve for (x^*, λ^*) directly. For example, sparse indefinite factorization methods such as MA27 (Duff and Reid [14]), MA57 (Duff [13]), and PARDISO (Schenk and Gärtner [39]) are chosen by some modern optimization solvers. These methods can be expensive when matrix H is large and dense.

- Range space method: This is also known as the *Schur complement method*. We can eliminate x^* from (1.7) and get $(AH^{-1}A^T)\lambda^* = -AH^{-1}c - b$, which we solve for λ^*. The value of x^* is then recovered by solving $Hx^* = -A^T\lambda^* - c$. This method requires the nonsingularity of H and expensive operations associated with forming and solving

$AH^{-1}A^T$ when m is not small.

- Null space method: Let $Z \in \Re^{n\times(n-m)}$ be the null space basis of A (i.e. $AZ = 0$) and $x^* = A^T x_A + Z x_Z$, where x_A is the solution of $(AA^T)x_A = b$. It is then easy to see that $Ax^* = b$, and x_Z solves the reduced problem

$$(Z^T H Z)x_Z = -Z^T(HA^T x_A + c). \tag{1.8}$$

Linear system of equations (1.8) can be solved by any approach mentioned in the previous section, especially the iterative method presented in Algorithm 1.1. We note that this method may require us to form the null space basis Z and the reduced Hessian $Z^T H Z$. Later in this thesis we will present CG algorithms for (1.6) that does not require explicit knowledge of Z.

1.3 General Nonlinear Optimization

1.3.1 Problem Statement

We are interested in solving the following general nonlinear program (NLP) problem

$$\begin{aligned} \underset{x\in\Re^n}{\text{minimize}} \quad & f(x) && \text{(1.9a)}\\ \text{subject to} \quad & c_E(x) = 0 && \text{(1.9b)}\\ & c_I(x) \geq 0, && \text{(1.9c)} \end{aligned}$$

where $f(x) : \Re^n \to \Re$, $c_E(x) : \Re^n \to \Re^t$ and $c_I(x) : \Re^n \to \Re^m$ are smooth functions. We assume that $t < n$. Problem (1.9) is of particular interest when the number of variables n, and possibly also the number of constraints t and m are large.

1.3.2 Constraint Qualification Conditions

Let $\mathcal{E}$ and $\mathcal{I}$ be the sets of indices of equality constraints and of inequality constraints, respectively. At any feasible set x of problem (1.9), we define the *active set* $\mathcal{A}(x)$ to be the union of set $\mathcal{E}$ with the indices of the active inequality constraints at point x. That is,

$$\mathcal{A}(x) \stackrel{\text{def}}{=} \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(x) = 0\}.$$

The following two definitions are the Linear Independence Constraint Qualification (LICQ) and Mangasarian-Fromovitz Constraint Qualification (MFCQ) in nonlinear programming:

Definition 1.2 *(LICQ) At given point x, LICQ holds if and only if the set of active constraint gradients $\{\nabla c_i(x) \mid i \in \mathcal{A}(x)\}$ is linearly independent.*

Definition 1.3 *(MFCQ) At given point x, MFCQ holds if and only if there exists a vector $d \in \Re^n$ such that*

$$\begin{aligned} \nabla c_i(x)^T d &> 0 \qquad \text{for all } i \in \mathcal{A}(x) \cap \mathcal{I}, \\ \nabla c_i(x)^T d &= 0 \qquad \text{for all } i \in \mathcal{E}, \end{aligned}$$

and the set of equality constraint gradients $\{\nabla c_i(x) \mid i \in \mathcal{E}\}$ is linearly independent.

1.4 Interior Point Algorithms

Active set methods and interior point methods are two main classes of methods for solving (1.9). During the step computation stage, active set methods guess the set of active constraints (known as the working set, which contains all the equality constraints and all the active inequality constraints) at the current iterate, then a step is computed by enforcing

the constraints in the working set as equalities.

Interior point methods, on the other hand, do not guess which inequality constraints are active. Instead, they introduce nonnegative slack variables $s \in \Re^m$ associated with all the inequality constraints, and then use a barrier term to keep the slack variables away from their bounds $s \geq 0$. A step in the (x, s) space is then computed using a primal or primal-dual approach. We will give a more detailed description of a specific interior point algorithm in this section.

1.4.1 Step Computation

Standard interior point methods for (1.9) (see Nocedal and Wright [33], Gould, Orban and Toint [23] and references therein) introduce non-negative slack variables $s \in \Re^m$ and formulate a series of barrier problems of the form

$$\underset{x \in \Re^n, s \in \Re^m}{\text{minimize}} \quad f(x) - \mu \sum_{i=1}^{m} \ln s_i \tag{1.10a}$$

$$\text{subject to} \quad c_{_E}(x) = 0 \tag{1.10b}$$

$$c_{_I}(x) - s = 0, \tag{1.10c}$$

where $\mu \to 0$ is the penalty parameter. The Lagrangian for the barrier problem (1.10) is given by

$$\mathcal{L}(x, s, \lambda_{_E}, \lambda_{_I}) = f(x) - \mu \sum_{i=1}^{m} \ln s_i - {\lambda_{_E}}^T c_{_E}(x) - {\lambda_{_I}}^T (c_{_I}(x) - s), \tag{1.11}$$

where $\lambda_{_E} \in \Re^t$ and $\lambda_{_I} \in \Re^m$ are the Lagrange multipliers of the barrier problem (1.10).

According to the basic ideas of Sequential Quadratic Programming (SQP) approaches, there are two ways for us to derive a linear system of equations, which can be solved for a step:

The first is to construct the following local quadratic programming model for the barrier problem (1.10):

$$\underset{d_x \in \Re^n, d_s \in \Re^m}{\text{minimize}} \quad \tfrac{1}{2} d_x^T (\nabla^2_{xx} \mathcal{L}) d_x + \tfrac{1}{2} d_s^T \Sigma d_s + \nabla f^T d_x - \mu (S^{-1} e)^T d_s \tag{1.12a}$$

$$\text{subject to} \quad A_{_E} d_x + c_{_E} = 0 \tag{1.12b}$$

$$A_{_I} d_x - d_s + (c_{_I} - s) = 0. \tag{1.12c}$$

Here the terms $\nabla^2_{xx} \mathcal{L} \in \Re^{n \times n}$ is the *Hessian of Lagrangian*, and matrix $\Sigma \in \Re^{m \times m}$, known as the *Hessian of barrier terms*, is diagonal and defined to be

$$\Sigma = S^{-1} \Lambda_{_I}, \quad \text{where } S = \operatorname{diag}\{s_i\}_{i=1}^m, \Lambda_{_I} = \operatorname{diag}\{(\lambda_{_I})_i\}_{i=1}^m. \tag{1.13}$$

Obviously the SQP problem (1.12) is a special case of the EQP problem (1.6). Therefore the first order necessary conditions of the quadratic programming subproblem (1.12) give a linear system of equations

$$\begin{bmatrix} \nabla^2_{xx} \mathcal{L} & 0 & A_{_E}{}^T & A_{_I}{}^T \\ 0 & \Sigma & 0 & -I \\ A_{_E} & 0 & 0 & 0 \\ A_{_I} & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ \lambda_{_E}^+ \\ \lambda_{_I}^+ \end{bmatrix} = \begin{bmatrix} -\nabla f \\ \mu S^{-1} e \\ -c_{_E} \\ -(c_{_I} - s) \end{bmatrix}. \tag{1.14}$$

The second derivation works directly with the first order necessary conditions of the

barrier problem (1.10):

$$\nabla f(x) - A_E(x)^T \lambda_E - A_I(x)^T \lambda_I = 0 \tag{1.15a}$$

$$-\mu S^{-1} e + \lambda_I = 0 \tag{1.15b}$$

$$c_E(x) = 0 \tag{1.15c}$$

$$c_I(x) - s = 0. \tag{1.15d}$$

If Newton's method is applied to the nonlinear system (1.15), we see that the resulting linear system of equations is also (1.14).

When the nonlinear programming problem does not have any equality constraints, linear system (1.14) reduces to

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 & A_I^T \\ 0 & \Sigma & -I \\ A_I & -I & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ \lambda_I^+ \end{bmatrix} = \begin{bmatrix} -\nabla f \\ \mu S^{-1} e \\ -(c_I - s) \end{bmatrix}. \tag{1.16}$$

In order to ensure global convergence of the algorithm and the positivity of the slack variables, we have to consider globalization techniques such as trust region and/or additional constraints such as the fraction to the boundary rule. In each iteration of the interior point algorithm we solve the quadratic programming subproblem (1.12) or the nonlinear system (1.15) governed by possible additional constraints for a step, during which the solution of linear system (1.14) or (1.16) will be the main computation.

Note that the solution of (1.14) or (1.16) is needed in every iteration of the interior point algorithm. When the problem is large, a lot of computation effort is spent on solving linear systems. Therefore it is important that we solve these linear equations efficiently.

1.4.2 Solving Linear Systems Arising in Interior Methods

It is not difficult to see that (1.14) and (1.16) are both special cases of the general KKT system (1.7). Hence any method we have mentioned for solving (1.7) can be applied to solve (1.14) or (1.16). However, recall that we are interested in solving large scale problems, so the size of these linear equations must be large. Thus some previously described methods may not work well for these specific equations. At the same time, due to the special structure of the coefficient matrices in (1.14) and (1.16), we may have more choices in choosing pivots to perform block eliminations when we solve these equations. These choices are based on a few important equivalences of (1.14) and (1.16), which we discuss below.

1. **Full system:** The original equation (1.14), without any block elimination, is often referred to as the *full system*. With definition

$$H = \begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 \\ 0 & \Sigma \end{bmatrix}, \quad A = \begin{bmatrix} A_E & 0 \\ A_I & -I \end{bmatrix}, \tag{1.17}$$

the problem has the same form as (1.7). Similarly, the full system for the problem without equality constraints is (1.16), which has the same form as (1.7) with

$$H = \begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 \\ 0 & \Sigma \end{bmatrix}, \quad A = \begin{bmatrix} A_I & -I \end{bmatrix}, \tag{1.18}$$

Because the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$ is usually large and sometimes not very sparse, its inverse is difficult to compute, which makes the range space method less attractive, as the range space method corresponds to a particular pivot choice, not necessarily the one that preserves sparsity the best. Instead, direct method and null space method are often used. For example, KNITRO/DIRECT (Waltz, Morales, Nocedal

and Orban [44]) implements the direct method and KNITRO/CG (Byrd, Hribar and Nocedal [7]) implements a null space method where the reduced problem is solved by an iterative approach.

2. **Semi-condensed system:** After eliminating d_s from the full system (1.14), we get the *semi-condensed system*:

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & {A_I}^T & {A_E}^T \\ A_I & -\Sigma^{-1} & 0 \\ A_E & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ \lambda_I^+ \\ \lambda_E^+ \end{bmatrix} = \begin{bmatrix} -\nabla f \\ \mu\Sigma^{-1}S^{-1}e - (c_I - s) \\ -c_E \end{bmatrix} \tag{1.19a}$$

$$d_s = \Sigma^{-1}\lambda_I^+ + \mu\Sigma^{-1}S^{-1}e. \tag{1.19b}$$

Another type of semi-condensed system is obtained by eliminating λ_I^+ from (1.14):

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & {A_I}^T\Sigma & {A_E}^T \\ \Sigma A_I & -\Sigma & 0 \\ A_E & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ \lambda_E^+ \end{bmatrix} = \begin{bmatrix} -\nabla f + \mu {A_I}^T S^{-1}e \\ -\Sigma(c_I - s) \\ -c_E \end{bmatrix} \tag{1.20a}$$

$$\lambda_I^+ = \Sigma d_s - \mu S^{-1}e. \tag{1.20b}$$

The semi-condensed systems without equality constraints are simpler in format:

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & {A_I}^T \\ A_I & -\Sigma^{-1} \end{bmatrix} \begin{bmatrix} d_x \\ \lambda_I^+ \end{bmatrix} = \begin{bmatrix} -\nabla f \\ \mu\Sigma^{-1}S^{-1}e - (c_I - s) \end{bmatrix} \tag{1.21a}$$

$$d_s = \Sigma^{-1}\lambda_I^+ + \mu\Sigma^{-1}S^{-1}e \tag{1.21b}$$

and

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & A_I{}^T\Sigma \\ \Sigma A_I & -\Sigma \end{bmatrix} \begin{bmatrix} d_x \\ d_s \end{bmatrix} = \begin{bmatrix} -\nabla f + \mu A_I{}^T S^{-1} e \\ -\Sigma(c_I - s) \end{bmatrix} \tag{1.22a}$$

$$\lambda_I^+ = \Sigma d_s - \mu S^{-1} e. \tag{1.22b}$$

With proper definitions of H and A, we see that equations (1.19a) and (1.20a) have the same form as (1.6), so they can be solved by the same techniques such as full space method and null space method with iterative solvers for the reduced problem. However, equations without equality constraints (1.21a) and (1.22a) have received more attention, although they do not have the format of the EQP problem (1.6). We will explain this in a slightly more general setting later in this section.

3. **Condensed system:** Further eliminating d_s or λ_I^+ from the semi-condensed system (1.19) and (1.20) respectively, we have the *condensed system*:

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} + A_I{}^T\Sigma A_I & A_E{}^T \\ A_E & 0 \end{bmatrix} \begin{bmatrix} d_x \\ \lambda_E^+ \end{bmatrix} = \begin{bmatrix} -\nabla f + \mu A_I{}^T S^{-1} e - A_I{}^T\Sigma(c_I - s) \\ -c_E \end{bmatrix} \tag{1.23a}$$

$$d_s = A_I d_x + (c_I - s) \tag{1.23b}$$

$$\lambda_I^+ = \Sigma d_s - \mu S^{-1} e. \tag{1.23c}$$

Without equality constraints, the condensed system is

$$(\nabla^2_{xx}\mathcal{L} + A_I{}^T\Sigma A_I)d_x = -\nabla f + \mu A_I{}^T S^{-1}e - A_I{}^T\Sigma(c_I - s) \tag{1.24a}$$

$$d_s = A_I d_x + (c_I - s) \tag{1.24b}$$

$$\lambda_I^+ = \Sigma d_s - \mu S^{-1}e. \tag{1.24c}$$

Matrix $\nabla^2_{xx}\mathcal{L} + A_I{}^T\Sigma A_I$ is normally referred to as the *condensed Hessian*. Again, we point out that (1.23a) has the EQP form and can be solved by direct method or null space method, although the term $A_I{}^T\Sigma A_I$ may be dense and expensive to form for nonlinear programming problems with general constraints. For example, software IPOPT (see Wächter [42] and Wächter and Biegler [43]) works with the condensed system (1.23) and implements a null space approach with the natural null space basis. Equation (1.24a) is an unconstrained linear system, and so can be solved by any linear algebra solver, especially the iterative approaches such as CG.

4. **Parametric augmented system:** We follow the idea proposed in Forsgren, Gill and Griffin [17] and consider a parametric point of view for some of the equivalent forms of the linear systems arising in interior point methods. The following family of equations is equivalent to (1.14):

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} + (1+\sigma)A_I{}^T\Sigma A_I & \sigma A_I{}^T & A_E{}^T \\ \sigma A_I & \sigma\Sigma^{-1} & 0 \\ A_E & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ -\lambda_I^+ \\ \lambda_E^+ \end{bmatrix} = \begin{bmatrix} b_1(\sigma) \\ b_2(\sigma) \\ -c_E \end{bmatrix} \tag{1.25a}$$

$$d_s = A_I d_x + (c_I - s), \tag{1.25b}$$

where $b_1(\sigma) = (1+\sigma)A_I^T(\mu S^{-1}e - \Sigma(c_I - s)) - \nabla f$ and $b_2(\sigma) = \mu\sigma\Sigma^{-1}S^{-1}e - \sigma(c_I - s)$. It is easy to see that when $\sigma = -1$, equation (1.25) becomes the semi-condensed system (1.19). When $\sigma = 0$, equation (1.25) reduces to the condensed system (1.23), where λ_I^+ is computed by (1.23c). When $\sigma = 1$, (1.25) is called the *doubly augmented system*, and takes the form

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} + 2A_I^T\Sigma A_I & A_I^T & A_E^T \\ A_I & \Sigma^{-1} & 0 \\ A_E & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ -\lambda_I^+ \\ \lambda_E^+ \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ -c_E \end{bmatrix} \tag{1.26a}$$

$$d_s = A_I d_x + (c_I - s), \tag{1.26b}$$

where the right hand side vectors are $b_1 = -\nabla f + 2A_I^T(\mu S^{-1}e - \Sigma(c_I - s))$ and $b_2 = \mu\Sigma^{-1}S^{-1}e - (c_I - s)$.

We note that the coefficient matrix in (1.25) or (1.26) is indefinite because of the existence of the zero matrix in the $(3,3)$ block. However, when there are no equality constraints in the problem, the parametric augmented system becomes

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} + (1+\sigma)A_I^T\Sigma A_I & \sigma A_I^T \\ \sigma A_I & \sigma\Sigma^{-1} \end{bmatrix} \begin{bmatrix} d_x \\ -\lambda_I^+ \end{bmatrix} = \begin{bmatrix} b_1(\sigma) \\ b_2(\sigma) \end{bmatrix} \tag{1.27a}$$

$$d_s = A_I d_x + (c_I - s), \tag{1.27b}$$

where $b_1(\sigma)$ and $b_2(\sigma)$ are defined above. When σ takes value at -1 or 0, the parametric augmented system reduces to the semi-condensed system (1.21) or the condensed system (1.24), respectively. Especially when $\sigma = 1$, the doubly augmented system

without equality constraints has the form

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} + 2A_I{}^T\Sigma A_I & A_I{}^T \\ A_I & \Sigma^{-1} \end{bmatrix} \begin{bmatrix} d_x \\ -\lambda_I^+ \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{1.28a}$$

$$d_s = A_I d_x + (c_I - s). \tag{1.28b}$$

We define $B(\sigma)$ to be the coefficient matrix in (1.27).Then it can be shown that

$$\text{inertia}(B(\sigma)) = \text{inertia}(\nabla^2_{xx}\mathcal{L} + A_I{}^T\Sigma A_I) + \text{inertia}(\sigma\Sigma^{-1}). \tag{1.29}$$

Considering the fact that the inertia of $\sigma\Sigma^{-1}$ is determined by the sign of σ because Σ is diagonal and positive definite, we know from (1.29) that when $\nabla^2_{xx}\mathcal{L} + A_I{}^T\Sigma A_I$ is positive definite,

$$\text{inertia}(B(\sigma)) = \begin{cases} (n, m, 0) & \text{if } \sigma < 0 \\ (n, 0, m) & \text{if } \sigma = 0 \\ (n + m, 0, 0) & \text{if } \sigma > 0. \end{cases} \tag{1.30}$$

In other words, when the condensed Hessian is positive definite, the parametric augmented system is also positive definite for any $\sigma > 0$. In this case we can choose to directly or iteratively solve any member of the family by fixing parameter σ, especially the doubly augmented system. Although in a larger space, it is positive definite for all the variables and λ_I^+ does not have to be recovered by extra calculations, therefore iterative methods like CG may work well for these equations.

5. **Augmented normal system:** Lu, Monteiro and O'Neal [31] assumes that the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$ has the eigenvalue factorization $\nabla^2_{xx}\mathcal{L} = VEV^T$, where $E \in \Re^{l\times l}$

is a diagonal matrix composed of the nonzero eigenvalues of $\nabla^2_{xx}\mathcal{L}$. Let $d_z = EV^T d_x$, then (1.14) is equivalent to

$$\begin{bmatrix} A_E(A_I{}^T\Sigma A_I)^{-1}A_E{}^T & A_E(A_I{}^T\Sigma A_I)^{-1}V \\ V^T(A_I{}^T\Sigma A_I)^{-1}A_E{}^T & V^T(A_I{}^T\Sigma A_I)^{-1}V + E^{-1} \end{bmatrix} \begin{bmatrix} \lambda_E^+ \\ d_z \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{1.31a}$$

$$d_x = (A_I{}^T\Sigma A_I)^{-1}v_3 \tag{1.31b}$$

$$d_s = A_I d_x + (c_I - s) \tag{1.31c}$$

$$\lambda_I^+ = \Sigma d_s - \mu S^{-1}e, \tag{1.31d}$$

where

$$\begin{aligned} v_1 &= c_E - A_E(A_I{}^T\Sigma A_I)^{-1}\left(\nabla f - \mu A_I{}^T S^{-1}e + A_I{}^T\Sigma(c_I - s)\right) \\ v_2 &= V^T(A_I{}^T\Sigma A_I)^{-1}\left(-\nabla f + \mu A_I{}^T S^{-1}e - A_I{}^T\Sigma(c_I - s)\right) \\ v_3 &= -A_E{}^T\lambda_E^+ - Vd_z - \nabla f + \mu A_I{}^T S^{-1}e - A_I{}^T\Sigma(c_I - s). \end{aligned}$$

Equation (1.31) is called the *augmented normal system*. It is important for us to note that the coefficient matrix in (1.31a) can be factorized into

$$\begin{bmatrix} A_E & 0 \\ V^T & I \end{bmatrix} \begin{bmatrix} (A_I{}^T\Sigma A_I)^{-1} & 0 \\ 0 & E^{-1} \end{bmatrix} \begin{bmatrix} A_E{}^T & V \\ 0 & I \end{bmatrix}. \tag{1.32}$$

To conclude this section, we point out the following while using the above equivalent systems to solve for a step in each iteration of the interior point algorithm:

- Every block elimination is equivalent to a Gaussian elimination to the full system (1.14) or (1.16) with a pre-defined order of pivots. In our derivation of the semi-condensed system and condensed system, matrices Σ and I are selected as pivots.

- For the lack of proper pivots, the displacement of the multipliers associated with the equality constraints λ_E^+ cannot be eliminated from the linear systems. In other words, λ_E^+ will always have to be solved together with d_x, as long as equality constraints are present in the nonlinear programming problem. However, if a small regularization term is added to the $(3,3)$ block of linear system (1.14), that is, if we consider to solve

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 & A_E{}^T & A_I{}^T \\ 0 & \Sigma & 0 & -I \\ A_E & 0 & -\gamma I & 0 \\ A_I & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ \lambda_E^+ \\ \lambda_I^+ \end{bmatrix} = \begin{bmatrix} -\nabla f \\ \mu S^{-1} e \\ -c_E \\ -(c_I - s) \end{bmatrix}, \tag{1.33}$$

where γ is a small regularization parameter, λ_E^+ can be eliminated and the resulting system is fully decoupled, which has the form

$$\begin{aligned} (\nabla^2_{xx}\mathcal{L} + \tfrac{1}{\gamma} A_E{}^T A_E + A_I{}^T \Sigma A_I) d_x &= v \\ d_s &= A_I d_x + (c_I - s) \\ \lambda_E^+ &= \tfrac{1}{\gamma}(A_E d_x + c_E) \\ \lambda_I^+ &= \Sigma d_s - \mu S^{-1} e, \end{aligned}$$

where $v = -\nabla f - \frac{1}{\gamma} A_E{}^T c_E - A_I{}^T \Sigma (c_I - s) + \mu A_I{}^T S^{-1} e$. However, the solution of system (1.33) does not satisfy the last two equations of (1.14). We will discuss this issue in a later chapter.

- When the nonlinear programming problem does not have equality constraints, the condensed system corresponds to a null space approach applied to (1.16) with a specific choice of the null space basis Z of the matrix A defined in (1.18). In detail, given the

definition of H and A in (1.18), the *natural null space basis* is defined as

$$Z = \begin{bmatrix} -I \\ A_I \end{bmatrix}. \tag{1.34}$$

With this null space basis, it is easy to verify that the reduced Hessian $Z^T H Z = \nabla^2_{xx}\mathcal{L} + A_I{}^T \Sigma A_I$, which is the coefficient matrix in (1.24a). In other words, the condensed Hessian $\nabla^2_{xx}\mathcal{L} + A_I{}^T \Sigma A_I$ is a specific reduced Hessian. Further more, when equality constraints are not present in the problem, the full system (1.16) has $n + m$ positive eigenvalues, m negative eigenvalues and no zero eigenvalues when $\nabla^2_{xx}\mathcal{L} + A_I{}^T \Sigma A_I$ is positive definite.

- The work of solving the condensed system and the parametric augmented system is affected by the sparsity of the condensed Hessian. Even if matrices $\nabla^2_{xx}\mathcal{L}$ and A_I are both sparse and Σ is diagonal, the $n \times n$ condensed Hessian $\nabla^2_{xx}\mathcal{L} + A_I{}^T \Sigma A_I$ may be dense (for example, when A_I has a dense row). However, if the nonlinear programming problem is a bound constrained problem, the rows of matrix A_I are all coordinate vectors, and it is easy to show that $A_I{}^T \Sigma A_I$ is also a diagonal matrix. In this case the condensed Hessian is easy to form.

- Solving the augmented normal system will not be efficient unless the eigenvalue factorization of $\nabla^2_{xx}\mathcal{L}$ can be performed at a reasonable cost and $A_I{}^T \Sigma A_I$ can be formed and inverted easily.

Chapter 2

Preconditioning

2.1 Concept of Preconditioning

Preconditioning can be explained well in the context of solving a square linear system of equations (1.1). We have mentioned that Krylov space methods are widely used iterative methods when solving (1.1). However, the convergence of Krylov space methods heavily depends on the condition number of the coefficient matrix A. If the condition number of A is large, Krylov space methods tend to converge slowly.

In order to accelerate the iterative algorithm, we consider an equivalent linear system of equations

$$P^{-1}Ax = P^{-1}b, \tag{2.1}$$

where $P \in \Re^{n \times n}$ is an invertible matrix called a *preconditioner*. We apply the Krylov space method to (2.1) instead of the original problem (1.1). It is easy to see that (1.1) and (2.1) have the same solution in exact arithmetic. Instead of (1.1), we solve (2.1) with the hope that the condition number of $P^{-1}A$ is better than that of A. Obviously when $P = A$, $P^{-1}A$ becomes the identity matrix and (2.1) gives the exact solution x^*. Hence $P = A$ is called

the *perfect preconditioner*. Generally speaking, matrix P should be an approximation to the coefficient matrix A. The "closer" P is to A, the better preconditioning effect we should expect from using this preconditioner.

Another way to view preconditioning is to transform the space in which the unpreconditioned iterative algorithm works. In other words, we introduce an invertible linear change of variables, hoping that the equivalence of the original problem (1.1) has a better condition number in the transformed space. Then we use an unpreconditioned iterative algorithm to solve the transformed problem, and retrieve the solution in the original space. We will explain this point of view in later sections.

We point out that the concept of preconditioning applies when we solve linear system of equations of specific forms, such as when A is symmetric and positive definite in system (1.1). Particularly, the convergence rate of the CG algorithm is related to the eigenvalue distribution of the coefficient matrix A. In this case the idea of preconditioning can be explained by the following: Finding a symmetric and positive definite matrix P such that $P^{-1}A$ has clustered eigenvalues. Obviously when $P = A$, we know that $P^{-1}A = I$, which has exactly one cluster of eigenvalues, and according to the discussion in Section 1.1, the CG algorithm should find the solution in 1 iteration in exact arithmetic.

For the optimization problem (1.2), preconditioning can be explained in a similar way: Finding a proper change of variables and transforming the space in which the unpreconditioned CG algorithm works, such that the second-order gradient of the transformed problem has a better condition number or a better eigenvalue distribution.

2.2 Preconditioned CG for Unconstrained Optimization

2.2.1 The Preconditioned CG Algorithm

We introduce linear change of variables $x = C^{-1}\hat{x}$ or $\hat{x} = Cx$ to problem (1.1) or (1.2), and apply unpreconditioned CG Algorithm 1.1 to the transformed problem. Let $M = C^T C$ or $M^{-1} = C^{-1}C^{-T}$. After some algebra we have the preconditioned CG algorithm for (1.1), shown in Algorithm 2.1.

Algorithm 2.1 *(Preconditioned CG algorithm for* (1.1)*) Choose* x_0*, compute* $r_0 = Ax_0 - b$ *and* $y_0 = M^{-1}r_0$*, define* $p_0 = -y_0$*,* $k = 0$*. Repeat until convergence*

$$\alpha_k = (r_k^T y_k)/(p_k^T A p_k) \tag{2.2a}$$

$$x_{k+1} = x_k + \alpha_k p_k \tag{2.2b}$$

$$r_{k+1} = r_k + \alpha_k A p_k \tag{2.2c}$$

$$y_{k+1} = M^{-1} r_{k+1} \tag{2.2d}$$

$$\beta_{k+1} = (r_{k+1}^T y_{k+1})/(r_k^T y_k) \tag{2.2e}$$

$$p_{k+1} = -y_{k+1} + \beta_{k+1} p_k \tag{2.2f}$$

$$k = k + 1. \tag{2.2g}$$

From another point of view, matrix M can be seen as a positive definite approximation of A. Better numerical performance is expected for a "good" approximation of A. Algorithm 2.1 reduces to unpreconditioned CG when $M = I$. Note that matrix C does not explicitly appear in Algorithm 2.1, and the preconditioning step (2.2d) requires us to solve a linear system of equations with coefficient matrix M.

The preconditioning step in Algorithm 2.1 is (2.2d), where we are interested in finding vector y_{k+1}. More generally speaking, matrix M may not even be defined explicitly, in which case the preconditioning step becomes $y_{k+1} = \mathcal{P}(r_{k+1})$, where $\mathcal{P}$ is an operator that maps r_{k+1} to y_{k+1}.

2.2.2 Preconditioning CG with a Trust Region

Let us consider the solution of a quadratic programming problem (1.4) governed by a trust region constraint.

We have discussed in Section 1.1.2 that we can introduce a change of variables $\hat{x} = Bx$, which changes the shape of the trust region from elliptical into circular for a general nonsingular matrix B. It is important for us to realize that this change of variables does not necessarily change the eigenvalue distribution of A in our favor. In other words, in problem (1.5) the eigenvalue distribution of matrix $B^{-T}AB^{-1}$ may not be better than that of A. Hence we may need another change of variables $\tilde{x} = C\hat{x}$ in order to do preconditioning. With this additional change of variables, the problem becomes

$$\underset{\tilde{x}\in\Re^n}{\text{minimize}} \quad \tilde{q}(\tilde{x}) = \tfrac{1}{2}\tilde{x}^T(C^{-T}B^{-T}AB^{-1}C^{-1})\tilde{x} - (C^{-T}B^{-T}b)^T\tilde{x} \tag{2.3a}$$

$$\text{subject to} \quad \left\|C^{-1}\tilde{x}\right\| \leq \Delta, \tag{2.3b}$$

which is equivalent to (1.4). Note that the trust region (2.3b) becomes elliptical again, in which case there are yet no convenient rules on when to terminate the CG iterations.

One may argue that we can solve instead

$$\underset{\tilde{x}\in\Re^n}{\text{minimize}} \quad \tilde{q}(\tilde{x}) = \tfrac{1}{2}\tilde{x}^T(C^{-T}B^{-T}AB^{-1}C^{-1})\tilde{x} - (C^{-T}B^{-T}b)^T\tilde{x} \tag{2.4a}$$

$$\text{subject to} \quad \|\tilde{x}\| \leq \Delta. \tag{2.4b}$$

We point out that problem (2.4) is not equivalent to (2.3), and therefore not equivalent to the original problem (1.4), because it has a different trust region constraint. In terms of the original variables x, the trust region of (2.4) is $\|CBx\| \leq \Delta$, which is obviously different from (1.4b) unless $C = I$. This indicates that the solution of (2.4) may or may not be seen as an approximation of the solution of the original problem (1.4).

2.3 Preconditioning General EQPs

2.3.1 Reduced Space Preconditioned CG Algorithm

We now present preconditioning ideas for the CG algorithms used in the null space methods when solving the equality constrained quadratic program (1.6).

As discussed in Section 1.2, we can use the null space approach to solve (1.6), where the expensive computation is the solution of the reduced problem (1.8). The following reduced space preconditioned CG algorithm is proposed in Golub and Van Loan [20], which is the application of Algorithm 2.1 on the reduced problem. For simplicity we denote $c_Z = Z^T(HAx_A + c)$ and let P_Z be the given preconditioner to Z^THZ.

Algorithm 2.2 *(Reduced space preconditioned CG algorithm for* (1.8)*) Choose an initial* x_Z*, compute* $r_Z = Z^THZx_Z + c_Z$ *and* $y_Z = {P_Z}^{-1}r_Z$*, define* $p_Z = -y_Z$*. Repeat until convergence*

$$\alpha = ({r_Z}^T y_Z)/({p_Z}^T Z^T H Z p_Z) \tag{2.5a}$$

$$x_Z = x_Z + \alpha p_Z \tag{2.5b}$$

$${r_Z}^+ = r_Z + \alpha Z^T H Z p_Z \tag{2.5c}$$

$${y_Z}^+ = {P_Z}^{-1} {r_Z}^+ \tag{2.5d}$$

$$\beta = ({r_Z}^+)^T {y_Z}^+ / ({r_Z}^T y_Z) \tag{2.5e}$$

$$p_Z = -{y_Z}^+ + \beta p_Z \tag{2.5f}$$

$$y_Z = {y_Z}^+, \; r_Z = {r_Z}^+. \tag{2.5g}$$

The preconditioning step in Algorithm 2.2 is (2.5d). We note that this algorithm requires the explicit knowledge of the null space basis Z. After computing x_Z from Algorithm 2.2, the full space solution of problem (1.6) is given by $x = A^T x_A + Z x_Z$.

2.3.2 Constraint Preconditioners

In order to avoid the explicit use of the null space basis Z, Gould, Hribar and Nocedal in [21] consider particular preconditioners of the form $P_Z = Z^T G Z$, where $G \in \Re^{n \times n}$ is an approximation to H and must be positive definite in the null space of A. We may incorporate the multiplication by Z and the addition of $A^T x_A$ into the reduced space preconditioned CG algorithm, and expand the algorithm into $(n+m)$-dimensional space, which results in the projected-preconditioned CG method shown in Algorithm 2.3. Here we define auxiliary vector $v \in \Re^m$ and

$$P = \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix}. \tag{2.6}$$

Algorithm 2.3 *(Projected-preconditioned CG algorithm for* (1.6)*) Choose* x_0 *that satisfies*

$Ax_0 = b$, compute $r_0 = Hx_0 + c$ and $\begin{bmatrix} y_0 \\ v \end{bmatrix} = P^{-1} \begin{bmatrix} r_0 \\ 0 \end{bmatrix}$*, define $p_0 = -y_0$, $k = 0$. Repeat until convergence*

$$\alpha_k = (r_k^T y_k)/(p_k^T H p_k) \tag{2.7a}$$

$$x_{k+1} = x_k + \alpha_k p_k \tag{2.7b}$$

$$r_{k+1} = r_k + \alpha_k H p_k \tag{2.7c}$$

$$\begin{bmatrix} y_{k+1} \\ v \end{bmatrix} = P^{-1} \begin{bmatrix} r_{k+1} \\ 0 \end{bmatrix} \tag{2.7d}$$

$$\beta_{k+1} = (r_{k+1}^T y_{k+1})/(r_k^T y_k) \tag{2.7e}$$

$$p_{k+1} = -y_{k+1} + \beta_{k+1} p_k \tag{2.7f}$$

$$k = k + 1. \tag{2.7g}$$

In Algorithm 2.3, (2.7d) is known as the *projection-preconditioning step*, because it projects r_{k+1}, the residual of the problem, onto the null space of the constraints A. This can be shown by the fact that the second equation in (2.7d) is $Ay_{k+1} = 0$. In fact, y_{k+1} can be explicitly expressed as $y_{k+1} = Z(Z^T G Z)^{-1} Z^T r_{k+1}$, where Z is any null space basis of A, hence y_{k+1} is called the *projected residual* or *preconditioned residual* of the problem. It is seen that Algorithm 2.3 is independent of the choice of the null space basis Z, or it works for any choice of Z, and it does not require any explicit knowledge of Z.

Matrix G determines the projections onto the null space, and further determines other parts of the algorithm. Usually we expect G to be some approximation to H. In this sense, matrix P defined in (2.6) is not only a projection matrix, but also a preconditioner. When $G = I$, y_{k+1} is the orthogonal projection and the unpreconditioned residual. Preconditioners of this form are widely regarded as *constraint preconditioners* because they respect the

constraint Jacobian A of the original problem (1.6).

When applying projected-preconditioned CG algorithm to solve (1.6) and computing the preconditioned residual y_{k+1}, various methods can be used, depending on how matrix G is defined. In the following we analyze a few cases:

1. If the user provides an explicit matrix G or it is easy to generate an explicit approximation G to matrix H from the real application, we can directly factorize P and solve (2.7d) exactly. This is necessary because we must have $Ay_{k+1} = 0$, which implicitly requires that G should be sparse enough if the size of the problem is large. A possible approximation to H could be the diagonal matrix proposed in Roma [37].

2. If G is given in a way such that its inverse G^{-1} is not expensive to compute, we can use the Schur complement method to solve (2.7d). For example, if H is positive definite, it may be possible to perform an incomplete Cholesky factorization $H = LL^T + R$ and define $G = LL^T$. We do not have to multiply L and L^T explicitly to form G, instead we compute $G^{-1}w = L^{-T}L^{-1}w$ by two backsolves, for any vector w. Then (2.7d) is equivalent to

$$\begin{bmatrix} LL^T & A^T \\ 0 & AL^{-T}L^{-1}A^T \end{bmatrix} \begin{bmatrix} y_{k+1} \\ v \end{bmatrix} = \begin{bmatrix} r_{k+1} \\ AL^{-T}L^{-1}r_{k+1} \end{bmatrix}. \tag{2.8}$$

   Another example is when we have a limited memory quasi-Newton approximation to H^{-1}, which can be used as G^{-1}, then (2.7d) is equivalent to

$$\begin{bmatrix} I & G^{-1}A^T \\ 0 & AG^{-1}A^T \end{bmatrix} \begin{bmatrix} y_{k+1} \\ v \end{bmatrix} = \begin{bmatrix} G^{-1}r_{k+1} \\ AG^{-1}r_{k+1} \end{bmatrix}. \tag{2.9}$$

   We can solve (2.8) or (2.9) for v and recover the value of y_{k+1}.

3. In Dollar, Gould and Wathen [12] the so-called *implicit factorization constraint preconditioners* are considered, where matrix P in the form of (2.6) is not formed explicitly. Instead, it is assumed that P has the factorization

$$\begin{bmatrix} P_1 & A^T \\ P_2 & 0 \end{bmatrix} \begin{bmatrix} B_1 & B_2^T \\ B_2 & B_3 \end{bmatrix} \begin{bmatrix} P_1^T & P_2^T \\ A & 0 \end{bmatrix}.$$

   Specific choices of matrices P_1, P_2, B_1, B_2 and B_3 are discussed such that the three block structured matrices above are not expensive to form and invert. The preconditioned residual y_{k+1} is then computed by solving linear systems associated with these matrices.

We point out that the projected-preconditioned CG algorithm 2.3 is actually the expanded form of the preconditioned CG algorithm applied to the null space approach for solving (1.6). In the null space, the reduced space problem (1.8) is an unconstrained positive definite problem solved by CG with a positive definite preconditioner. Hence, the convergence rate of Algorithm 2.3 is determined by the eigenvalue distribution of $(Z^TGZ)^{-1}(Z^THZ)$ for any null space basis Z. See Keller, Gould and Wathen [28] for more theoretical analysis. Some people tend to think of Algorithm 2.3 as a full space indefinite preconditioner applied to a full space indefinite problem, which often prevents us from seeing the null space nature of the algorithm.

The projected-preconditioned CG algorithm implemented in practice may need some modification to Algorithm 2.3. In KNITRO/CG, a projected-preconditioned CG algorithm with a circular trust region on the step is implemented, where Steihaug's rule [40] is adopted in case the search step is too long or negative curvature is detected.

2.3.3 Non-Constraint Preconditioners

Besides constraint preconditioners (2.6), some researchers have proposed other preconditioning ideas for the iterative method when solving (1.6).

Coleman and Verma in [9] follow the reduced space preconditioned CG framework 2.2 and consider a specific reduced space preconditioner. When we solve (1.8) using Algorithm 2.2, the preconditioning is performed by (2.5d). [9] proposes to use preconditioners of the form $P_z = \tilde{Z}^T \tilde{H} \tilde{Z}$, where $\tilde{H}$ is a positive definite approximation of H, and $\tilde{Z}$ is an approximation of Z. The main concern when we choose $\tilde{Z}$ is that we should avoid forming $\tilde{Z}^T \tilde{H} \tilde{Z}$ explicitly.

In order to do so, we suppose that $\tilde{A}$ is an approximation of A such that $\tilde{A}$ can be partitioned into $\tilde{A} = [\tilde{A}_1 \quad \tilde{A}_2]$ where $\tilde{A}_1$ is nonsingular. The natural null space basis of $\tilde{A}$ is then given by

$$\tilde{Z} = \begin{bmatrix} -\tilde{A}_1^{-1} \tilde{A}_2 \\ I \end{bmatrix}, \tag{2.10}$$

which has two good properties:

1. For any vector $r_z \in \Re^{n-m}$,
$$\tilde{Z}^T \begin{bmatrix} 0 \\ r_z \end{bmatrix} = r_z. \tag{2.11}$$

2. For any vector $w \in \Re^{n-m}$,
$$\tilde{Z} w = \begin{bmatrix} -\tilde{A}_1^{-1} \tilde{A}_2 w \\ w \end{bmatrix}. \tag{2.12}$$

   In other words, the last $n - m$ elements of $\tilde{Z}w$ form vector w itself.

With matrix $\tilde{Z}$ defined by (2.10), we perform the preconditioning step $y_z^+ = (\tilde{Z}^T \tilde{H} \tilde{Z})^{-1} r_z^+$

in Algorithm 2.2 by first solving

$$\begin{bmatrix} \tilde{H} & \tilde{A}^T \\ \tilde{A} & 0 \end{bmatrix} \begin{bmatrix} y \\ u \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0 \\ r_Z{}^+ \end{bmatrix} \\ 0 \end{bmatrix} \tag{2.13}$$

for $y \in \Re^n$, and then letting $y_Z{}^+$ be the vector formed by the last $n-m$ elements of y (Note that these procedures do not require $\tilde{Z}$ and $\tilde{Z}^T\tilde{H}\tilde{Z}$ explicitly). This can be explained by noticing the solution y of (2.13) can be expressed by

$$y = \tilde{Z}(\tilde{Z}^T\tilde{H}\tilde{Z})^{-1}\tilde{Z}^T \begin{bmatrix} 0 \\ r_Z{}^+ \end{bmatrix} = \tilde{Z}(\tilde{Z}^T\tilde{H}\tilde{Z})^{-1} r_Z{}^+ = \tilde{Z} y_Z{}^+.$$

The second equality holds because of (2.11). According to (2.12), the last $n-m$ elements of $y = \tilde{Z} y_Z{}^+$ form vector $y_Z{}^+$ itself.

Comparing with the projected-preconditioned CG algorithm, this reduced space preconditioned CG has its own pros and cons: The advantage is that we have freedom in choosing $\tilde{A}$, so we have control on the sparsity of the preconditioner in (2.13), while in Algorithm 2.3 we must use the exact matrix A in (2.6). The disadvantage is that this reduced space algorithm is all at the null space level only except the preconditioning step, thus the exact null space Z is still needed in the computation, while Algorithm 2.3 is a null space method completely expanded in the full space and it is independent of Z.

2.4 Preconditioning the Linear Systems Arising in Interior Methods

In this section we discuss the preconditioning issues when we use the CG algorithm to solve the linear system of equations and their equivalences arising in each iteration of an interior point method.

2.4.1 Sources of Ill-conditioning

For later references, we point out three possible sources of ill-conditioning in these linear systems, and note that not all of them will necessarily appear in a specific linear system to be solved.

- Ill-conditioning in the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$: This is caused by the poor eigenvalue distribution of $\nabla^2_{xx}\mathcal{L}$.

- Ill-conditioning in the Hessian of barrier terms Σ: From (1.15b), we have that for a specific inequality constraint $(c_I)_i$, $s_i(\lambda_I)_i \approx \mu$ near the solution of the barrier problem with barrier parameter μ. According to the definition (1.13) of matrix Σ, the corresponding diagonal element of Σ is $s_i^{-1}(\lambda_I)_i \approx \mu/(s_i)^2$. We consider two cases:

  - If $(c_I)_i$ is active, then $s_i \longrightarrow 0$, so $\mu/(s_i)^2 \longrightarrow +\infty$ for any fixed μ;
  - If $(c_I)_i$ is inactive, then $\mu/(s_i)^2 \longrightarrow 0$ as $\mu \longrightarrow 0$.

  Therefore we conclude that some diagonal elements of Σ will diverge to infinity for each barrier problem, and some diagonal elements will shrink to zero as the iterates are approaching the solution of the nonlinear programming problem. This obviously indicates that matrix Σ has widely spread eigenvalues and becomes more and more ill-conditioned, which is unfavorable to CG.

- Ill-conditioning in the constraints: Matrices A_E and A_I could also be ill-conditioned. However, if we consider constraint preconditioners of the form (2.6), the ill-conditioning in the constraints will be removed.

In this thesis we will focus on the implementation of constraint preconditioners applied to the linear systems arising in interior methods. In the following we will discuss the preconditioning issues for different types of these linear systems classified in 1.4.2.

2.4.2 Preconditioning the Full System

The full system (1.14) and (1.16) both suffer from the ill-conditioning in the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$ and the Hessian of barrier terms Σ, no matter whether equality constraints are present in the problem. The projection-preconditioning step in the projected-preconditioned CG algorithm has the form

$$\begin{bmatrix} P_x & 0 & {A_E}^T & {A_I}^T \\ 0 & P_s & 0 & -I \\ A_E & 0 & 0 & 0 \\ A_I & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} y_x \\ y_s \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} r_x \\ r_s \\ 0 \\ 0 \end{bmatrix}, \tag{2.14}$$

when equality constraints are present, or

$$\begin{bmatrix} P_x & 0 & {A_I}^T \\ 0 & P_s & -I \\ A_I & -I & 0 \end{bmatrix} \begin{bmatrix} y_x \\ y_s \\ v_2 \end{bmatrix} = \begin{bmatrix} r_x \\ r_s \\ 0 \end{bmatrix}, \tag{2.15}$$

when equality constraints are not present.

We note, again, that the projected-preconditioned CG algorithm is in fact a specific preconditioned CG algorithm used in the null space approach. Hence we should target

on improving the eigenvalue distribution of the reduced Hessian. However, analyzing the reduced Hessian itself is difficult because most of the time the null space basis is not explicitly formed and operated. Therefore, we will only consider the Hessian matrices defined in (1.17)-(1.18).

Strategies Used to Perform Block Eliminations

In terms of solving for the preconditioned residual y_x and y_s from (2.14)-(2.15), there are two main strategies: We can either solve these equations without any block elimination (which requires P_x and P_s to be explicitly available), or solve a certain equivalent form of these equations after some block elimination (where P_x and P_s are not necessarily required to be explicitly available). The most useful block elimination takes the form

$$\begin{bmatrix} P_x + {A_I}^T P_s A_I & {A_E}^T \\ A_E & 0 \end{bmatrix} \begin{bmatrix} y_x \\ v_1 \end{bmatrix} = \begin{bmatrix} r_x + {A_I}^T r_s \\ 0 \end{bmatrix}, \quad y_s = A_I y_x \tag{2.16}$$

for (2.14) or

$$(P_x + {A_I}^T P_s A_I) y_x = r_x + {A_I}^T r_s, \quad y_s = A_I y_x \tag{2.17}$$

for (2.15). In these equations we can form matrix $P_x + {A_I}^T P_s A_I$ instead of P_x and P_s individually.

Later in this chapter we will see that the equations above share the same format as the constraint preconditioners (2.21)-(2.23) for the condensed form of the linear systems in interior point methods. In fact, in an interior point framework without trust regions and in exact arithmetic, we can show that the following two methods are equivalent:

1. Form the linear system as (1.14)-(1.16), use preconditioners of the form (2.14)-(2.15), and solve for y_x and y_s using (2.16)-(2.17). That is, the "condensing" (block elimina-

tion) happens at the projection-preconditioning level.

2. Form the linear system as (1.23)-(1.24), and solve them using constraint preconditioners (2.21)-(2.23). That is, the "condensing" (block elimination) happens at the level of linear equations to be solved.

Nevertheless, the two methods above are slightly different in a CG framework with trust regions. The trust region for the CG algorithm in the first method is on (d_x, d_s), while the trust region for the CG algorithm in the second method is on d_x only. If the trust region bound is reached by some iterate during the CG process and special stopping rules (such as Steihaug's rule) truncate the search step, the two methods above may give different total steps.

In the following we will talk about using the first method to solve (2.14)-(2.15), where P_x and P_s are explicitly available. We leave the discussion on forming $P_x + A_I^T P_s A_I$ in (2.16)-(2.17) to the section of preconditioning the condensed system, while keeping in mind the difference between the two methods mentioned above.

Preconditioning Σ

We note that Σ is a diagonal matrix, therefore it is natural to use a diagonal matrix P_s to approximate it. As a result, P_s^{-1} is diagonal and cheap to form as well. A few possible choices of P_s are:

1. Perfect barrier preconditioner: $P_s = \Sigma = S^{-1}\Lambda_I$. This choice is such that $P_s^{-1}\Sigma = I$, which has only one cluster of eigenvalues at 1.

2. Approximately perfect barrier preconditioner: $P_s = \mu S^{-2}$. This is an approximation to the exact barrier preconditioner, because we know from (1.15b) that $\Lambda_I \approx \mu S^{-1}$ near the solution of each barrier problem, and then $\Sigma = S^{-1}\Lambda_I \approx S^{-1}(\mu S^{-1}) = \mu S^{-2}$. With

this choice $P_s^{-1}\Sigma = \frac{1}{\mu}S^2\Sigma = \frac{1}{\mu}S\Lambda_I \approx \frac{1}{\mu}(\mu I) = I$. Hence the preconditioned matrix has some eigenvalues tightly clustered at 1.

3. Slack scaling barrier preconditioner: $P_s = S^{-2}$. This is the choice in Byrd, Hribar and Nocedal [7]. With this choice $P_s^{-1}\Sigma = S\Lambda_I \approx \mu I$. Thus the preconditioned matrix has some small eigenvalues tightly clustered at 0.

Preconditioning $\nabla^2_{xx}\mathcal{L}$

1. If a matrix P_x is given by the user or easy to be generated to approximate $\nabla^2_{xx}\mathcal{L}$ (for example, P_x is a diagonal matrix), we can directly factorize the coefficient matrix in (2.14) or (2.15) and solve for y_x and y_s exactly. This is necessary because we must have

$$\begin{bmatrix} A_E & 0 \\ A_I & -I \end{bmatrix} \begin{bmatrix} y_x \\ y_s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

for problems with equality constraints or $A_I y_x - y_s = 0$ for problems without equality constraints.

2. If P_x is defined such that P_x^{-1} is easy to compute, and we know that P_s^{-1} is easy to compute as well, we can use the Schur complement approach to solve (2.14) or (2.15). For example, if P_x^{-1} is defined by a limited memory quasi-Newton approach, we can solve for the preconditioned residual y_x and y_s by (2.9), where

$$G = \begin{bmatrix} P_x & 0 \\ 0 & P_s \end{bmatrix}, \quad A = \begin{bmatrix} A_E & 0 \\ A_I & -I \end{bmatrix}$$

for problems with equality constraints or

$$G = \begin{bmatrix} P_x & 0 \\ 0 & P_s \end{bmatrix}, \quad A = \begin{bmatrix} A_I & -I \end{bmatrix}$$

for problems without equality constraints. We point out that an incomplete Cholesky approximation of $\nabla^2_{xx}\mathcal{L}$ may not be a good choice for P_x because $\nabla^2_{xx}\mathcal{L}$ is not necessary positive definite, so its incomplete Cholesky factorization may not exist. However, if we know $\nabla^2_{xx}\mathcal{L}$ is positive definite, we can first find matrix L such that $\nabla^2_{xx}\mathcal{L} \approx LL^T$ and define $P_x = LL^T$ without explicit multiplication, then use the Schur complement approach to do preconditioning, since $AG^{-1}A^T = (L^{-1}A^T)^T(L^{-1}A^T)$ and $L^{-1}A^T$ can be formed by some backsolves.

2.4.3 Preconditioning the Semi-condensed System

In this section we discuss the preconditioning issues when using CG to solve the semi-condensed system (1.19)-(1.22). We observe that in (1.19) and (1.21) the Hessian of barrier terms appears as Σ^{-1}, which is still diagonal, and still has some elements diverging to infinity for each barrier problem and some elements shrinking to zero near the solution of the nonlinear programming problem, according to our discussion in Section 2.4.1.

When Equality Constraints Are Present

When equality constraints are present in the nonlinear programming problem, the semi-condensed system is (1.19) or (1.20). For either of them, the coefficient matrix has the form of the EQP (1.7), and suffers from the ill-conditioning in both the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$ and the Hessian of the barrier terms Σ.

For (1.19), the projection-preconditioning step in the projected-preconditioned CG algo-

rithm with a constraint preconditioner will be

$$\begin{bmatrix} P_x & A_I^T & A_E^T \\ A_I & -P_s^{-1} & 0 \\ A_E & 0 & 0 \end{bmatrix} \begin{bmatrix} y_x \\ y_I \\ v_1 \end{bmatrix} = \begin{bmatrix} r_x \\ r_I \\ 0 \end{bmatrix}. \tag{2.18}$$

Similarly, for (1.20), the projection-preconditioning step will be

$$\begin{bmatrix} P_x & A_I^T P_s & A_E^T \\ P_s A_I & -P_s & 0 \\ A_E & 0 & 0 \end{bmatrix} \begin{bmatrix} y_x \\ y_s \\ v_1 \end{bmatrix} = \begin{bmatrix} r_x \\ r_s \\ 0 \end{bmatrix}. \tag{2.19}$$

We have the same three choices for matrix P_s in (2.18)-(2.19) as in the discussion on the full system (see Section 2.4.2). However, in order to compute the preconditioned residual efficiently, we must have an explicit form of P_x and perform a direct factorization to the coefficient matrices in (2.18) and (2.19). The Schur complement method in Section 2.4.2 no longer works because the 2×2 leading submatrices of (2.18)-(2.19) do not have the block diagonal form, so their inverse matrices are expensive to form and operate.

When Equality Constraints Are Not Present

When equality constraints are not present in the nonlinear programming problem, the semi-condensed system (1.21)-(1.22) both suffer from the ill-conditioning in the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$ and the Hessian of the barrier terms Σ.

Equations (1.21)-(1.22) are no longer of the form (1.7), so the projected-preconditioned CG algorithm does not work. However, we can still use a preconditioned iterative approach, such as Algorithm 2.1, to solve these equations. Bergamaschi, Gondzio and Zilli in [3] consider such preconditioned iterative methods when $\nabla^2_{xx}\mathcal{L}$ is positive definite. They propose

to construct a diagonal approximation to $\nabla^2_{xx}\mathcal{L}$ and form preconditioners of the form

$$P_D = \begin{bmatrix} D & A_I^T \\ A_I & -\Sigma^{-1} \end{bmatrix} \tag{2.20}$$

for (1.21), where $D = \text{diag}\,\{\nabla^2_{xx}\mathcal{L}\}$. They also prove that when (2.20) is used to precondition (1.21), the eigenvalues of the preconditioned matrix

$$P_D{}^{-1} \begin{bmatrix} \nabla^2_{xx}\mathcal{L} & A_I^T \\ A_I & -\Sigma^{-1} \end{bmatrix}$$

are all distributed between $\min\{1, \lambda_{\min}(D^{-1}\nabla^2_{xx}\mathcal{L})\}$ and $\max\{1, \lambda_{\max}(D^{-1}\nabla^2_{xx}\mathcal{L})\}$. Especially, at least m eigenvalues of the preconditioned matrix are 1.

Similar preconditioners of the form (2.20) are also possible. For example, D can be defined as a limited memory quasi-Newton approximation to $\nabla^2_{xx}\mathcal{L}$.

After a preconditioner in the form of (2.20) is constructed, we factorize it directly and perform backsolves. We mention that incomplete Cholesky factorization of $\nabla^2_{xx}\mathcal{L}$ is not an efficient choice for D in this case, even if $\nabla^2_{xx}\mathcal{L}$ is known to be positive definite, since the Schur complement approach is expensive to implement, given the nonzero $(2,2)$ blocks in (1.21)-(1.22).

2.4.4 Preconditioning the Condensed System

We consider the preconditioning issues when solving the condensed system (1.23)-(1.24) in this section. We point out that these equations may not individually suffer from either the ill-conditioning in the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$, or the Hessian of barrier terms Σ. Instead, the source of ill-conditioning is the poor eigenvalue distribution of the condensed

Hessian $\nabla^2_{xx}\mathcal{L} + {A_I}^T \Sigma A_I$. The ill-conditioning in $\nabla^2_{xx}\mathcal{L}$ or Σ may be shaded by the matrix multiplication or summation.

Here we assume that the condensed Hessian can be formed efficiently.

When Equality Constraints Are Present

When there are equality constraints in the nonlinear programming problem, the condensed system (1.23) has the form of an EQP (1.7). Therefore if the condensed Hessian is formulated, we an use any constraint preconditioned iterative method described in Section 2.3.2 to solve (1.23). The projection-preconditioning step is

$$\begin{bmatrix} P_x + {A_I}^T P_s A_I & {A_E}^T \\ A_E & 0 \end{bmatrix} \begin{bmatrix} y_x \\ v_1 \end{bmatrix} = \begin{bmatrix} r_x \\ 0 \end{bmatrix}. \tag{2.21}$$

Obviously it is natural to use a diagonal matrix P_s as an approximation of Σ. So we have the same three choices for P_s as the ones mentioned in Section 2.4.2. We summarize the ideas in the following, depending on how P_x is given:

1. If P_x is given by the user or easy to be generated, we can formulate $P_x + {A_I}^T P_s A_I$ explicitly and factorize it directly to solve for y_x. This is necessary because we must have $A_E y_x = 0$.

2. If $(P_x + {A_I}^T P_s A_I)^{-1}$ is easy to be computed, then we can use the Schur complement approach to compute y_x. For example, we can apply incomplete Cholesky factorization to $\nabla^2_{xx}\mathcal{L} + {A_I}^T \Sigma A_I$ and get $\nabla^2_{xx}\mathcal{L} + {A_I}^T \Sigma A_I = LL^T + R$ (note that it is not required that $\nabla^2_{xx}\mathcal{L}$ to be positive definite, but that $\nabla^2_{xx}\mathcal{L} + {A_I}^T \Sigma A_I$ is positive definite, which is a weaker requirement since ${A_I}^T \Sigma A_I$ is positive definite), then we define $P_x + {A_I}^T P_s A_I =$

LL^T. We have from (2.21) that

$$\begin{bmatrix} LL^T & {A_E}^T \\ 0 & (L^{-1}{A_E}^T)^T(L^{-1}{A_E}^T) \end{bmatrix} \begin{bmatrix} y_x \\ v_1 \end{bmatrix} = \begin{bmatrix} r_x \\ L^{-T}L^{-1}r_x \end{bmatrix},$$

from the second equation of which we solve for v_1 and recover the value of y_x.

3. One possible way for us to avoid using the Schur complement approach is to add a small regularization term to the $(2,2)$ block of (2.21), as we discussed in Section 1.4.2. The resulting system is

$$\begin{bmatrix} P_x + {A_I}^T P_s A_I & {A_E}^T \\ A_E & -\gamma I \end{bmatrix} \begin{bmatrix} y_x \\ v_1 \end{bmatrix} = \begin{bmatrix} r_x \\ 0 \end{bmatrix},$$

which is equivalent to

$$(P_x + \tfrac{1}{\gamma}{A_E}^T A_E + {A_I}^T P_s A_I) y_x = r_x \tag{2.22a}$$

$$v_1 = \tfrac{1}{\gamma} A_E y_x. \tag{2.22b}$$

Then we can compute y_x from (2.22a) if $(P_x + \frac{1}{\gamma}{A_E}^T A_E + {A_I}^T P_s A_I)^{-1}$ is easy to be computed. For example, we can apply incomplete Cholesky factorization to $\nabla^2_{xx}\mathcal{L} + \frac{1}{\gamma}{A_E}^T A_E + {A_I}^T \Sigma A_I$ and get $\nabla^2_{xx}\mathcal{L} + \frac{1}{\gamma}{A_E}^T A_E + {A_I}^T \Sigma A_I = LL^T + R$, then define $P_x + \frac{1}{\gamma}{A_E}^T A_E + {A_I}^T P_s A_I = LL^T$ and backsolve y_x from (2.22). One important thing to remember is that the vector y_x computed this way does not satisfy $A_E y_x = 0$, the second equation of (2.21). We can use an additional projection step and solve

$$\begin{bmatrix} I & {A_E}^T \\ A_E & 0 \end{bmatrix} \begin{bmatrix} \bar{y}_x \\ v \end{bmatrix} = \begin{bmatrix} y_x \\ 0 \end{bmatrix}$$

for $\bar{y}_x$, and return $\bar{y}_x$ as the preconditioned residual in the projected CG algorithm. This approach, because of the difficulty to determine a proper regularization parameter γ and the extra workload to compute ${A_E}^T A_E$, is not very promising.

We also point out that the regularization technique described above can also be used to avoid the Schur complement approach when preconditioning the full system. If we add regularization term $-\gamma I$ to the $(3,3)$ block of (2.14) and perform eliminations, the coefficient matrix for computing y_x will be the same as that in (2.22a). Again, we need an additional projection step to ensure that the preconditioned residual is in the null space of

$$\begin{bmatrix} A_E & 0 \\ A_I & -I \end{bmatrix}.$$

Instead of the projected-preconditioned CG algorithm, in Wächter [42] a null space approach for the condensed system (1.23) is implemented with the natural null space basis Z of the equality constraints. That is, the overall reduced Hessian has the form $Z^T(\nabla^2_{xx}\mathcal{L} + {A_I}^T\Sigma A_I)Z$ and is solved by a preconditioned CG algorithm. Preconditioning issues for this CG algorithm are discussed and several ideas are tested. The first idea is to use a limited memory BFGS for $Z^T(\nabla^2_{xx}\mathcal{L} + {A_I}^T\Sigma A_I)Z$, following the idea in Morales and Nocedal [32]. The limited memory quasi-Newton preconditioner for the CG algorithm is obtained by the information from the CG iterations in the previous interior point iteration. The second idea is to build a quasi-Newton approximation $\tilde{P} \approx Z^T\nabla^2_{xx}\mathcal{L}Z$ and use $\tilde{P} + Z^T{A_I}^T\Sigma A_I Z$ as the preconditioner for the overall reduced Hessian.

When Equality Constraints Are Not Present

When equality constraints are nor present in the nonlinear programming problem, the condensed system reduces to (1.24). Note that (1.24a) is an unconstrained linear system of

equations whose coefficient matrix is the condensed Hessian, which we can use the preconditioned CG approach presented in Algorithm 2.1 to solve. The preconditioned step then takes the form

$$(P_x + A_I^T P_s A_I) y_x = r_x. \tag{2.23}$$

Again we mention that the three choices of P_s are given in Section 2.4.2. We consider the following cases:

1. If P_x is given or $(P_x + A_I^T P_s A_I)^{-1}$ is easy to be computed, we can solve (2.23) for y_x. In [24] Hei, Nocedal and Waltz consider the solution of bound constrained optimization problems, where $A_I^T \Sigma A_I$ is diagonal. The exact condensed Hessian is formed explicitly and incompletely Cholesky factorized as $\nabla^2_{xx}\mathcal{L} + A_I^T \Sigma A_I = LL^T + R$. The preconditioner is defined as $P_x + A_I^T P_s A_I = LL^T$, and we backsolve $Lw = r_x$ and $L^T y_x = w$ to compute y_x. Without any modification, this approach can be generalized to the problems with only inequality constraints, provided that $A_I^T \Sigma A_I$ is not very dense. Johnson and Sofer [27] also studies the bound constrained problems, where a preconditioned CG algorithm with a diagonal preconditioner is applied to (1.24a). The diagonal elements of the preconditioner are the same as those of $\nabla^2_{xx}\mathcal{L} + A_I^T \Sigma A_I$, which amounts to use a diagonal approximation for $\nabla^2_{xx}\mathcal{L}$ and use the $A_I^T \Sigma A_I$ term exactly. The diagonal preconditioner is then easily inverted to compute y_x. This approach enjoys the nice property of the image reconstruction application that the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$, which has its own special structure, does not have to be formed explicitly and the matrix-vector product $(\nabla^2_{xx}\mathcal{L})v$ can be computed at a reasonable cost. The preconditioner can also be formed by only computing the diagonal elements of $\nabla^2_{xx}\mathcal{L}$ instead of the whole matrix.

   When the optimization problem has general nonlinear constraints other than bounds, forming and inverting $P_x + A_I^T P_s A_I$ may be prohibitively expensive, even if P_s is

diagonal (for example, $P_s = \Sigma$). This can shown by the fact that

$$A_I^T P_s A_I = \sum_{j=1}^{m} (P_s)_j a_j^T a_j, \tag{2.24}$$

where a_j is the j-th row of A_I and $(P_s)_j$ is the j-th diagonal element of P_s. The following observations may prevent us from forming and inverting $P_x + A_I^T P_s A_I$ efficiently. First, even if A_I only has one dense row, the matrix $A_I^T P_s A_I$ will be dense, so the whole matrix $P_x + A_I^T P_s A_I$ will be dense, no matter P_x is sparse or dense. Second, when m is large, there are too many terms in the right hand side of (2.24), so computing $A_I^T P_s A_I$ will be expensive.

Hei, Nocedal and Waltz in [25] propose to properly drop terms from the right hand side of (2.24) and reduce the workload to form matrix $A_I^T P_s A_I$. We drop a term $(P_s)_j a_j^T a_j$ from (2.24) if the following two conditions are both satisfied:

- The density of a_j (that is, the percentage of nonzero elements in a_j) is greater than a certain threshold value;
- The diagonal element $(P_s)_j$ is less than a certain threshold value.

The first condition aims at reducing the number of terms caused by the dense rows of A_I, but it does not necessarily always achieve this goal. The second condition is aligned with the preconditioning idea in Forsgren, Gill and Griffin [17], where we think small terms of $(P_s)_j$ do not make great contributions to (2.24) and can be safely dropped.

2. We recall that the preconditioning step in Algorithm 2.1 does not necessarily require an explicit preconditioner matrix. Instead it only requires the image of a mapping from r_k. A possible approach is to solve equation $(\nabla^2_{xx}\mathcal{L} + A_I^T \Sigma A_I)w = r_k$ approximately and regard the approximate solution vector as y_k, which also can be considered as

preconditioning.

2.4.5 Preconditioning the Parametric Augmented System

In Section 1.4.2 we have expressed some equivalent forms of the linear systems arising in interior point methods by a family of linear equations (1.25) characterized by a single parameter σ. In this section we will study the preconditioning issues when solving (1.27), under the assumption that there are no equality constraints and ${A_I}^T \Sigma A_I$ can be formed. We note that the doubly augmented system (1.28) is a special member of the family, so preconditioning the doubly augmented system becomes a special case of preconditioning the family of linear systems. The discussion in this section is based on Forsgren, Gill and Griffin [17].

Constraint preconditioners of the form

$$P(\sigma) = \begin{bmatrix} M + (1+\sigma){A_I}^T \Sigma A_I & \sigma {A_I}^T \\ \sigma A_I & \sigma \Sigma^{-1} \end{bmatrix} \tag{2.25}$$

are considered for the iterative algorithm for solving equation (1.27a), where M is a symmetric approximation to $\nabla^2_{xx}\mathcal{L}$ such that $M + {A_I}^T \Sigma A_I$ is positive definite. Similarly to (1.29)-(1.30), we have

$$\text{inertia}(P(\sigma)) = \begin{cases} (n, m, 0) & \text{if } \sigma < 0 \\ (n, 0, m) & \text{if } \sigma = 0 \\ (n+m, 0, 0) & \text{if } \sigma > 0. \end{cases} \tag{2.26}$$

Therefore $P(\sigma)$ is positive definite when $\sigma > 0$. Especially when $\sigma = 1$, this positive definite preconditioner can be applied to the doubly augmented system, which is also positive definite when $\nabla^2_{xx}\mathcal{L} + {A_I}^T \Sigma A_I$ is positive definite. When $\sigma = 0$, the full matrix (2.25) is not positive

definite but $M + A_I{}^T \Sigma A_I$ is positive definite and can be applied to the condensed system (1.24a) only in the variables d_x.

Additionally, [17] constructs two so-called *active set preconditioners* for the doubly augmented system ($\sigma = 1$). We notice that some elements of the diagonal matrix Σ tend to be very large (those corresponding to the active constraints) and the rest of them converge to zero (those corresponding to inactive constraints), when the iterations are close to the solution of each barrier problem. The difference in the size of these diagonal elements is one of the sources of the ill-conditioning. Without loss of generality we denote

$$\Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}, \quad A_I = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \tag{2.27}$$

where Σ_1 is made of the large elements of Σ, Σ_2 is made of all the rest diagonal elements of Σ, and matrix A_I is partitioned accordingly. Then (2.25) becomes

$$P(\sigma) = \begin{bmatrix} M + (1+\sigma)A_1^T\Sigma_1 A_1 + (1+\sigma)A_2^T\Sigma_2 A_2 & \sigma A_1^T & \sigma A_2^T \\ \sigma A_1 & \sigma\Sigma_1^{-1} & 0 \\ \sigma A_2 & 0 & \sigma\Sigma_2^{-1} \end{bmatrix}. \tag{2.28}$$

The two active set preconditioners considered by [17] are:

$$P_1(\sigma) = \begin{bmatrix} M + (1+\sigma)A_1^T\Sigma_1 A_1 + \sigma A_2^T\Sigma_2 A_2 & \sigma A_1^T & \sigma A_2^T \\ \sigma A_1 & \sigma\Sigma_1^{-1} & 0 \\ \sigma A_2 & 0 & \sigma\Sigma_2^{-1} \end{bmatrix} \tag{2.29a}$$

$$P_2(\sigma) = \begin{bmatrix} M + (1+\sigma)A_1^T\Sigma_1 A_1 & \sigma A_1^T & 0 \\ \sigma A_1 & \sigma\Sigma_1^{-1} & 0 \\ 0 & 0 & \sigma\Sigma_2^{-1} \end{bmatrix}. \tag{2.29b}$$

Preconditioner (2.29a) can be seen as obtained by removing $A_2^T \Sigma_2 A_2$ from the $(1,1)$ block of (2.28). This is reasonable, because Σ_2 does not contain large elements of Σ and so $A_2^T \Sigma_2 A_2$ will not play an important role in the preconditioner. Preconditioner (2.29b) further removes all the terms related to A_2, so it is not exactly a constraint preconditioner, but is more attractive when A_I is expensive to form. Theoretical results show that $P_1(\sigma)$ and $P_2(\sigma)$ work almost as well as the constraint preconditioner $P(\sigma)$ defined by (2.28) when $\sigma > 0$.

2.4.6 Preconditioning the Augmented Normal System

Preconditioning ideas for the augmented normal system (1.31) can be motivated by the factorization (1.32) of the coefficient matrix in (1.31a). Since the matrix V comes from the eigenvalue factorization of $\nabla^2_{xx}\mathcal{L}$, the first and third matrices in (1.32) are readily available. The second matrix of (1.32) has a block diagonal structure, and the $(2,2)$ block itself is a diagonal matrix. It is therefore natural for us to construct a sparse and easily invertible approximation $M \approx {A_I}^T \Sigma A_I$, and design a preconditioner of the form

$$\begin{bmatrix} A_E & 0 \\ V^T & I \end{bmatrix} \begin{bmatrix} M^{-1} & 0 \\ 0 & E^{-1} \end{bmatrix} \begin{bmatrix} {A_E}^T & V \\ 0 & I \end{bmatrix} \tag{2.30}$$

for equation (1.31a). As mentioned in the case of the condensed system, forming and inverting M could be expensive (see (2.24) and the discussion there). A possible way to construct M is using the dropping rule mentioned for (2.24), according to the density of the rows of A_I and the size of the diagonal elements of Σ.

When the nonlinear programming problem is only bound constrained, however, matrix ${A_I}^T \Sigma A_I$ is diagonal and cheap to form, and it is natural to use $M = {A_I}^T \Sigma A_I$. Lu, Monteiro and O'Neal in [31] consider the bound constrained case and assume that matrix E is positive definite (i.e. $\nabla^2_{xx}\mathcal{L}$ is positive semidefinite). A preconditioner is designed by ranking the

diagonal elements of M^{-1} and E^{-1} and selecting the rows of the third matrix in (2.30) only corresponding to the large elements of these diagonal matrices.

2.5 Numerical Experiments

2.5.1 Numerical Experiments with a MATLAB Implementation

We develop MAKCG, a MATLAB interior point code that solves general nonlinear programming problems with constraints, following the framework of Byrd, Hribar and Nocedal [7]. The search step, a combination of a normal step and a tangential step, is the solution of a linear system of equations of the form (1.14), which is solved by the decomposition approach proposed in Byrd [5] and Omojokun [34]. Particularly, the tangential step is the solution of an equality constrained quadratic subproblem governed by a trust region, which we solve by the null space method. Although MAKCG does not contain all the features of KNITRO/CG, we believe that it is sufficiently robust and efficient to test the preconditioners for interior point methods.

Experiment Setup

Some test problems are chosen from the CUTEr collection (see Gould, Orban and Toint [22] and Bongartz, Conn, Gould and Toint [4]) using versions of the models formulated in Ampl (see Fourer, Gay and Kernighan [18]). Since the MAKCG code is not optimized for speed, we have chosen test problems with a relatively small number of variables. These test problems are classified into three categories: bound constrained problems, nonlinear programming problems with only inequality constraints and general nonlinear programming problems with both equality and inequality constraints.

We implement in MAKCG some of the preconditioning ideas mentioned in earlier sections

and compare the results with the unpreconditioned solvers. In the following we will report in each experiment the name of the problem (`problem`), the characteristics of the problem (the number of variables n, the number of equality constraints t and the number of inequality constraints m), the preconditioning options being used (`option`), the final objective function value (`final objective`), the number of iterations of the interior method (`#iteration`), the total number of CG iterations (`#total CG`), the average number of CG iterations per interior point iteration (`#average CG`) and the CPU time (`time`).

The preconditioning ideas tested are based on the full system formulation that we have discussed in Section 2.4.2. The preconditioning options are labeled as

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

where a indicates the way we solve for the preconditioned residual and it takes values at either 1 or 2. $a = 1$ means that we have P_x and P_s explicitly available and directly factorize (2.14)-(2.15) to obtain y_x and y_s, while $a = 2$ indicates that we do not have P_x and P_s explicitly, but we incompletely factorize $\nabla^2_{xx}\mathcal{L} + A_I{}^T \Sigma A_I = LL^T + R$ using the `cholinc` function in MATLAB with drop tolerance 0.001, define $P_x + A_I{}^T P_s A_I = LL^T$ and solve for y_x and y_s from (2.16)-(2.17). The values of b and c indicate the preconditioning options used for the Hessian of Lagrangian and the Hessian of barrier terms, respectively. Obviously b and c are meaningful only when $a = 1$. Different combinations of b and c give different preconditioners. For example, option $(1, 1, 2)$ means that we explicitly have $P_x = |\text{diag}\{\nabla^2_{xx}\mathcal{L}\}|$ and $P_s = \Sigma$, and then factorize (2.14)-(2.15) to solve for y_x and y_s. The current default in KNITRO/CG is option $(1, 0, 0)$.

The MAKCG experiment options are summarized in Table 2.1.

$a = 1$ (P_x and P_s explicitly known)	
$b = 0$	No Hessian preconditioning ($P_x = I$)
$b = 1$	Diagonal Hessian preconditioning ($P_x = \|\text{diag}\{\nabla^2_{xx}\mathcal{L}\}\|$)
$c = 0$	Slack scaling barrier preconditioning ($P_s = S^{-2}$)
$c = 1$	Approximately perfect barrier preconditioning ($P_s = \mu S^{-2}$)
$c = 2$	Perfect barrier preconditioning ($P_s = \Sigma$)
$a = 2$ ($P_x + A_I^T P_s A_I$ determined implicitly)	
$P_x + A_I^T P_s A_I = LL^T$ where $\nabla^2_{xx}\mathcal{L} + A_I^T \Sigma A_I = LL^T + R$	

Table 2.1: Preconditioning options for MAKCG.

Experiment Results and Observations

The results for some bound constrained problems, for some problems with only inequality constraints but not just bounds and for general problems with both equality and inequality constraints are listed in Table 2.2, Table 2.3 and Table 2.4, respectively. We have some interesting observations and basic conclusions:

1. We see that the choice of preconditioners does not greatly affect the number of interior point iterations in the set of bound constrained problems, except problem `cvxbqp1`. However, the number of interior point iterations is dramatically affected for most of the general nonlinear programming problem. We will discuss the deterioration in the number of interior point iterations later in more details.

2. If we compare the average number of CG iterations when only barrier preconditioners are used (i.e. options $(1, 0, 1)$ and $(1, 0, 2)$). As expected, we see that the average number of CG iterations is generally decreased with respect to the benchmark option $(1, 0, 0)$. Hence these barrier preconditioners can be considered as successful in this context. But it is not clear which barrier preconditioner is to be preferred.

3. The diagonal preconditioner for the Hessian of Lagrangian (i.e. options $(1, 1, *)$), although sometimes reduces the average number of CG iterations, only provides marginal

benefit. We then conclude that this type of preconditioners are of limited use.

4. The incomplete Cholesky implementation of the preconditioners (i.e. option $(2, -, -)$) is very successful. It almost always substantially reduce the average number of CG iterations. It also provides some benefit in terms of CPU time, but in a less effective manner. We explain this by the cost of performing the incomplete Cholesky factorizations, which is related to the density of the condensed Hessian, and the multiplication and re-factorization operations when solving (2.16) for problems with equality constraints.

Deterioration of Interior Point Iterations

Generally, one may expect that the preconditioners will not affect the number of interior point iterations dramatically, since it is simply a mechanism for accelerating the step computation procedure. Nevertheless, we see the deterioration of interior point iterations in many of our test problems. We analyze the implementation of the preconditioning ideas and conclude that the deterioration of the number of interior point iterations is due to two main reasons: the shape of the trust region and the barrier stop test.

For simplicity we assume that $P_x = I$, that is, there is no preconditioning for the Hessian of Lagrangian.

In our interior point implementation, there is a trust region constraint for the step in each iteration (see the description in Section 1.4.1). In more details, the trust region for the step computation has the form

$$\left\| \begin{bmatrix} d_x \\ S^{-1} d_s \end{bmatrix} \right\| \leq \Delta, \tag{2.31}$$

where Δ is the trust region radius for the step. The S^{-1} part of the trust region is often referred to as the affine scaling strategy, which is a standard approach to ensure the global

convergence of the interior method. See Byrd, Gilbert and Nocedal [6] for more details. Also note that all the three proposed preconditioners have at least S^{-1} in the form of P_s.

Meanwhile, we remember that preconditioning can be seen as a change of variables that transforms the original problem. We can see the standard choice, $P_s = S^{-2}$, corresponds to a change of variables of the form

$$\tilde{d}_x = d_x, \quad \tilde{d}_s = S^{-1} d_s. \tag{2.32}$$

This change of variables has its pros and cons: On one hand, it is easy to see that under (2.32) the trust region (2.31) happens to become circular and has form

$$\left\| \begin{bmatrix} \tilde{d}_x \\ \tilde{d}_s \end{bmatrix} \right\| \leq \Delta,$$

which makes the implementation of the projected-preconditioned CG algorithm very easy, since we know that Steihaug's rules described in Section 1.1.2 work well for circular trust regions. The success of KNITRO/CG indicates that trust regions of the form (2.31) control well the rate at which the slacks approach zero. Therefore we would like to respect the trust region. On the other hand, we have shown in Section 2.4.2 that the Hessian of barrier terms after the change of variables (2.32) is approximately μI, which has some eigenvalues tightly clustered at zero, increasing the ill-conditioning of the linear system.

This disadvantage motivates us to design other barrier preconditioners, among which we consider $P_s = \mu S^{-2}$. This choice corresponds to a change of variables of the form

$$\tilde{d}_x = d_x, \quad \tilde{d}_s = \sqrt{\mu} S^{-1} d_s. \tag{2.33}$$

Because of the lack of efficient stopping rules for CG algorithms with an elliptical trust

region, we have to enforce a circular trust region for the scaled variables $(\tilde{d}_x, \tilde{d}_s)$. That is, in terms of the original variables, the trust region that we actually enforce is

$$\left\| \begin{bmatrix} d_x \\ \sqrt{\mu} S^{-1} d_s \end{bmatrix} \right\| \leq \Delta. \tag{2.34}$$

Obviously this means that the desired trust region (2.31) is not respected. We have seen that the Hessian of barrier terms after (2.33) is approximately I. That is, the condition number of the linear system is improved. Nevertheless, when the barrier parameter μ is small, (2.34) does not penalize a step approaching the slack bounds as severely as (2.31). This allows the interior method to approach the boundary of the feasible region of the problem prematurely and produce very small steps, which explains the deterioration in the number of interior point iterations.

Another proposed barrier preconditioner $P_s = \Sigma = S^{-1}\Lambda_I$ can also be ineffective for similar reasons because we know $\Lambda_I \approx \mu S^{-1}$ near the solution of the barrier problem, so this choice of P_s is close to the one we have discussed above. Moreover, if the multiplier estimates λ_I are inaccurate, the trust region will not properly control d_s.

problem (n,t,m)	option	final objective	#iteration	#total CG	#average CG	time (sec)
biggsb1	(1,0,0)	$+1.5015971301e-02$	31	3962	$1.278e+02$	$3.138e+01$
(100,0,198)	(1,0,1)	$+1.5015971301e-02$	29	2324	$8.014e+01$	$1.931e+01$
	(1,0,2)	$+1.5015971301e-02$	28	2232	$7.971e+01$	$1.848e+01$
	(1,1,0)	$+1.5015971301e-02$	30	3694	$1.231e+02$	$2.813e+01$
	(1,1,1)	$+1.5015971301e-02$	30	2313	$7.710e+01$	$1.858e+01$
	(1,1,2)	$+1.5015971301e-02$	30	2241	$7.470e+01$	$1.876e+01$
	(2,–,–)	$+1.5015971301e-02$	31	44	$1.419e+00$	$1.950e+00$
cvxbqp1	(1,0,0)	$+9.0450040000e+02$	11	91	$8.273e+00$	$4.360e+00$
(200,0,400)	(1,0,1)	$+9.0453998374e+02$	8	112	$1.400e+01$	$4.240e+00$
	(1,0,2)	$+9.0450040000e+02$	53	54	$1.019e+00$	$1.146e+01$
	(1,1,0)	$+9.0454000245e+02$	30	52	$1.733e+00$	$9.130e+00$
	(1,1,1)	$+9.0450040000e+02$	30	50	$1.667e+00$	$9.400e+00$
	(1,1,2)	$+9.0454001402e+02$	47	48	$1.021e+00$	$1.330e+01$
	(2,–,–)	$+9.0450040000e+02$	11	18	$1.636e+00$	$2.480e+00$
jnlbrng1	(1,0,0)	$-1.7984674056e-01$	29	5239	$1.807e+02$	$8.500e+01$
(324,0,324)	(1,0,1)	$-1.7984674056e-01$	27	885	$3.278e+01$	$1.999e+01$
	(1,0,2)	$-1.7984674056e-01$	29	908	$3.131e+01$	$2.016e+01$
	(1,1,0)	$-1.7984674056e-01$	29	5082	$1.752e+02$	$9.563e+01$
	(1,1,1)	$-1.7984674056e-01$	27	753	$2.789e+01$	$2.940e+01$
	(1,1,2)	$-1.7988019171e-01$	26	677	$2.604e+01$	$2.702e+01$
	(2,–,–)	$-1.7984674056e-01$	30	71	$2.367e+00$	$6.780e+00$
obstclbm	(1,0,0)	$+5.9472925926e+00$	28	7900	$2.821e+02$	$1.931e+02$
(225,0,450)	(1,0,1)	$+5.9473012340e+00$	18	289	$1.606e+01$	$1.195e+01$
	(1,0,2)	$+5.9472925926e+00$	31	335	$1.081e+01$	$1.587e+01$
	(1,1,0)	$+5.9472925926e+00$	27	6477	$2.399e+02$	$1.391e+02$
	(1,1,1)	$+5.9472925926e+00$	29	380	$1.310e+01$	$1.998e+01$
	(1,1,2)	$+5.9473012340e+00$	18	197	$1.094e+01$	$1.154e+01$
	(2,–,–)	$+5.9472925926e+00$	27	49	$1.815e+00$	$7.050e+00$
pentdi	(1,0,0)	$-7.4969998494e-01$	27	260	$9.630e+00$	$6.460e+00$
(250,0,250)	(1,0,1)	$-7.4969998502e-01$	25	200	$8.000e+00$	$5.620e+00$
	(1,0,2)	$-7.4969998500e-01$	28	205	$7.321e+00$	$5.920e+00$
	(1,1,0)	$-7.4969998494e-01$	28	256	$9.143e+00$	$1.091e+01$
	(1,1,1)	$-7.4992499804e-01$	23	153	$6.652e+00$	$8.510e+00$
	(1,1,2)	$-7.4969998502e-01$	26	132	$5.077e+00$	$9.020e+00$
	(2,–,–)	$-7.4969998494e-01$	27	41	$1.519e+00$	$3.490e+00$
torsion1	(1,0,0)	$-4.8254023392e-01$	26	993	$3.819e+01$	$8.200e+00$
(100,0,200)	(1,0,1)	$-4.8254023392e-01$	25	298	$1.192e+01$	$4.020e+00$
	(1,0,2)	$-4.8254023392e-01$	24	274	$1.142e+01$	$3.660e+00$
	(1,1,0)	$-4.8254023392e-01$	26	989	$3.804e+01$	$9.390e+00$
	(1,1,1)	$-4.8254023392e-01$	25	274	$1.096e+01$	$3.980e+00$
	(1,1,2)	$-4.8254023392e-01$	25	250	$1.000e+01$	$3.770e+00$
	(2,–,–)	$-4.8254023392e-01$	25	52	$2.080e+00$	$1.720e+00$
torsionb	(1,0,0)	$-4.0993481087e-01$	25	1158	$4.632e+01$	$1.038e+01$
(100,0,200)	(1,0,1)	$-4.0993481087e-01$	25	303	$1.212e+01$	$4.050e+00$
	(1,0,2)	$-4.0993481087e-01$	23	282	$1.226e+01$	$3.710e+00$
	(1,1,0)	$-4.0993481087e-01$	25	1143	$4.572e+01$	$1.038e+01$
	(1,1,1)	$-4.0993481087e-01$	24	274	$1.142e+01$	$3.910e+00$
	(1,1,2)	$-4.0993481087e-01$	23	246	$1.070e+01$	$3.190e+00$
	(2,–,–)	$-4.0993481087e-01$	24	49	$2.042e+00$	$1.360e+00$

Table 2.2: MAKCG results for bound constrained problems.

problem (n,t,m)	option	final objective	#iteration	#total CG	#average CG	time (sec)
airport	(1,0,0)	$+4.7952710080e+04$	20	760	$3.800e+01$	$7.770e+00$
(84,0,210)	(1,0,1)	$+4.7952710204e+04$	18	412	$2.289e+01$	$4.780e+00$
	(1,0,2)	$+4.7952710203e+04$	18	234	$1.300e+01$	$3.290e+00$
	(1,1,0)	$+4.7952703365e+04$	88	631	$7.170e+00$	$1.157e+01$
	(1,1,1)	$+4.7952710080e+04$	24	133	$5.542e+00$	$2.940e+00$
	(1,1,2)	$+4.7952703360e+04$	34	92	$2.706e+00$	$3.280e+00$
	(2,–,–)	$+4.7952710202e+04$	14	29	$2.071e+00$	$1.080e+00$
chandheq	(1,0,0)	$+3.8074663778e-08$	23	332	$1.443e+01$	$3.330e+00$
(100,0,100)	(1,0,1)	$+3.8582052731e-08$	20	362	$1.810e+01$	$3.280e+00$
	(1,0,2)	$+3.8583031712e-08$	20	361	$1.805e+01$	$3.260e+00$
	(1,1,0)	$+5.0615075579e-09$	24	104	$4.333e+00$	$2.590e+00$
	(1,1,1)	$+3.8945183881e-08$	18	78	$4.333e+00$	$1.970e+00$
	(1,1,2)	$+3.8588499524e-08$	19	82	$4.316e+00$	$2.080e+00$
	(2,–,–)	$+3.8074663759e-08$	23	48	$2.087e+00$	$2.610e+00$
expfitc	(1,0,0)	$+2.3312955877e-02$	21	97	$4.619e+00$	$9.990e+00$
(5,0,501)	(1,0,1)	$+2.3304184127e-02$	33	162	$4.909e+00$	$1.619e+01$
	(1,0,2)	$+2.3304184127e-02$	41	168	$4.098e+00$	$1.838e+01$
	(1,1,0)	$+2.3310364663e-02$	38	170	$4.474e+00$	$1.746e+01$
	(1,1,1)	$+2.3310201248e-02$	272	1297	$4.768e+00$	$1.286e+02$
	(1,1,2)	$+2.3302627540e-02$	266	966	$3.632e+00$	$1.114e+02$
	(2,–,–)	$+2.3304184127e-02$	27	67	$2.481e+00$	$7.860e+00$
gpp	(1,0,0)	$+1.4400931122e+04$	17	4558	$2.681e+02$	$1.883e+02$
(250,0,498)	(1,0,1)	$+1.4400927187e+04$	22	489	$2.223e+01$	$2.996e+01$
	(1,0,2)	$+1.4400927347e+04$	29	933	$3.217e+01$	$5.001e+01$
	(1,1,0)	$+1.4400931122e+04$	21	978	$4.657e+01$	$5.118e+01$
	(1,1,1)	$+1.4400931111e+04$	32	321	$1.003e+01$	$3.193e+01$
	(1,1,2)	$+1.4400931121e+04$	21	70	$3.333e+00$	$1.587e+01$
	(2,–,–)	$+1.4400931122e+04$	17	211	$1.241e+01$	$2.142e+01$
himmelbi	(1,0,0)	$-1.7549999889e+03$	38	4142	$1.090e+02$	$2.125e+01$
(100,0,112)	(1,0,1)	$-1.7549999889e+03$	43	5167	$1.202e+02$	$2.644e+01$
	(1,0,2)	$-1.7549999976e+03$	47	4018	$8.549e+01$	$2.093e+01$
	(1,1,0)	$-1.7549999528e+03$	183	7088	$3.873e+01$	$4.063e+01$
	(1,1,1)	$-1.7549999949e+03$	42	5213	$1.241e+02$	$2.705e+01$
	(1,1,2)	$-1.7549999708e+03$	133	5516	$4.147e+01$	$3.121e+01$
	(2,–,–)	$-1.7549999890e+03$	48	659	$1.373e+01$	$3.330e+00$
ksip	(1,0,0)	$+5.7580344773e-01$	37	996	$2.692e+01$	$2.768e+02$
(20,0,1001)	(1,0,1)	$+5.7580344773e-01$	40	409	$1.022e+01$	$1.475e+02$
	(1,0,2)	$+5.7580344773e-01$	35	352	$1.006e+01$	$1.266e+02$
	(1,1,0)	$+5.7580344773e-01$	37	671	$1.814e+01$	$2.028e+02$
	(1,1,1)	$+5.7580344621e-01$	35	118	$3.371e+00$	$7.600e+01$
	(1,1,2)	$+5.7580344758e-01$	28	28	$1.000e+00$	$4.486e+01$
	(2,–,–)	$+5.7580344773e-01$	37	312	$8.432e+00$	$4.577e+01$

Table 2.3: MAKCG results for problems with only inequality constraints.

problem (n,t,m)	option	final objective	#iteration	#total CG	#average CG	time (sec)
dual1	(1,0,0)	$+3.5018630104e-02$	69	6056	$8.777e+01$	$4.914e+01$
(85,1,170)	(1,0,1)	$+3.5018629747e-02$	52	1854	$3.565e+01$	$1.695e+01$
	(1,0,2)	$+3.5018629747e-02$	42	1676	$3.990e+01$	$1.448e+01$
	(1,1,0)	$+3.5018630007e-02$	93	5637	$6.061e+01$	$4.736e+01$
	(1,1,1)	$+3.5020065859e-02$	59	1655	$2.805e+01$	$1.578e+01$
	(1,1,2)	$+3.5018629747e-02$	71	1712	$2.411e+01$	$1.731e+01$
	(2,–,–)	$+3.5018629747e-02$	62	154	$2.484e+00$	$5.240e+00$
hs119	(1,0,0)	$+2.4489970752e+02$	13	85	$6.538e+00$	$3.400e-01$
(16,8,32)	(1,0,1)	$+2.4489970752e+02$	12	70	$5.833e+00$	$3.100e-01$
	(1,0,2)	$+2.4489970752e+02$	18	88	$4.889e+00$	$4.100e-01$
	(1,1,0)	$+2.4489970752e+02$	26	78	$3.000e+00$	$4.400e-01$
	(1,1,1)	$+2.4489970752e+02$	28	74	$2.643e+00$	$4.500e-01$
	(1,1,2)	$+2.4489970752e+02$	25	55	$2.200e+00$	$3.800e-01$
	(2,–,–)	$+2.4489970752e+02$	13	22	$1.692e+00$	$1.800e-01$
linspanh	(1,0,0)	$-7.6999999996e+01$	11	188	$1.709e+01$	$1.780e+00$
(72,32,144)	(1,0,1)	$-7.7000000004e+01$	9	276	$3.067e+01$	$2.360e+00$
	(1,0,2)	$-7.7000000000e+01$	39	65	$1.667e+00$	$1.990e+00$
	(1,1,0)	$-7.7000000045e+01$	19	312	$1.642e+01$	$3.010e+00$
	(1,1,1)	$-7.7000000001e+01$	28	431	$1.539e+01$	$4.260e+00$
	(1,1,2)	$-7.7000000005e+01$	7	105	$1.500e+01$	$1.140e+00$
	(2,–,–)	$-7.7000000020e+01$	14	14	$1.000e+00$	$6.500e-01$
loadbal	(1,0,0)	$+4.5285331684e-01$	17	167	$9.824e+00$	$7.600e-01$
(31,11,62)	(1,0,1)	$+4.5285430574e-01$	69	735	$1.065e+01$	$3.040e+00$
	(1,0,2)	$+4.5285430593e-01$	71	763	$1.075e+01$	$3.160e+00$
	(1,1,0)	$+4.5285464232e-01$	27	269	$9.963e+00$	$1.240e+00$
	(1,1,1)	$+4.5285430588e-01$	63	646	$1.025e+01$	$2.810e+00$
	(1,1,2)	$+4.5285430588e-01$	47	482	$1.026e+01$	$2.120e+00$
	(2,–,–)	$+4.5285326581e-01$	17	20	$1.176e+00$	$3.200e-01$
model	(1,0,0)	$+5.7421652154e+03$	12	132	$1.100e+01$	$1.160e+00$
(60,26,126)	(1,0,1)	$+5.7421633528e+03$	14	137	$9.786e+00$	$1.310e+00$
	(1,0,2)	$+5.7421634994e+03$	9	41	$4.556e+00$	$5.200e-01$
	(1,1,0)	$+5.7421651307e+03$	12	123	$1.025e+01$	$1.170e+00$
	(1,1,1)	$+5.7421662179e+03$	8	60	$7.500e+00$	$6.900e-01$
	(1,1,2)	$+5.7421637049e+03$	8	29	$3.625e+00$	$5.300e-01$
	(2,–,–)	$+5.7257898047e+03$	15	24	$1.600e+00$	$5.400e-01$
portfl6	(1,0,0)	$+2.5794194264e-02$	15	120	$8.000e+00$	$3.500e-01$
(12,1,24)	(1,0,1)	$+2.5793715666e-02$	27	263	$9.741e+00$	$6.500e-01$
	(1,0,2)	$+2.5793715666e-02$	26	265	$1.019e+01$	$6.500e-01$
	(1,1,0)	$+2.5791918205e-02$	13	119	$9.154e+00$	$3.300e-01$
	(1,1,1)	$+2.5791918158e-02$	15	119	$7.933e+00$	$3.300e-01$
	(1,1,2)	$+2.5794194264e-02$	16	101	$6.312e+00$	$3.000e-01$
	(2,–,–)	$+2.5794194264e-02$	15	25	$1.667e+00$	$1.700e-01$

Table 2.4: MAKCG results for problems with both equality and inequality constraints.

2.5.2 Numerical Experiments with KNITRO/CG

Experiment Setup

We implement the most successful preconditioning idea so far, the incomplete Cholesky preconditioning, in KNITRO/CG and perform a series of numerical experiments. Therefore in this section we will only compare two versions of KNITRO/CG: the default version (without preconditioning) and the preconditioned version (with incomplete Cholesky preconditioning). For the latter, we factorize $\nabla^2_{xx}\mathcal{L} + {A_I}^T \Sigma A_I = LL^T + R$ using the incomplete Cholesky algorithm proposed by Lin and Moré [30], define $P_x + {A_I}^T P_s A_I = LL^T$ and solve for y_x and y_s from (2.16)-(2.17). We allow the user to choose the memory parameter `pmem`, which controls the amount of fill-in in the incomplete factorization. The most frequent used values of `pmem` ($0, 2, 5$ and 10) are tested in our experiments, among which 0 means that LL^T has the same sparse pattern as $\nabla^2_{xx}\mathcal{L} + {A_I}^T \Sigma A_I$, and larger values of this parameter indicates that LL^T is closer to the complete Cholesky factorization of $\nabla^2_{xx}\mathcal{L} + {A_I}^T \Sigma A_I$.

The test problems are still chosen from the CUTEr collection with the Ampl format. We test two classes of problems: bound constrained problems and problems with only inequality constraints. We exclude some of the repeated models from the bound constrained problems in the CUTEr set and select 29 problems for which the sizes can be large enough. For the general nonlinear programs, we select all the problems in the CUTEr set that do not have equality constraints, and whose inequality constraints are not just bounds.

Results for Bound Constrained Problems

We present the numerical results with some performance profiles. Figures 2.1-2.4 show the results on the 29 bound constrained problems in terms of number of iterations, number of function evaluations, average number of CG iterations and CPU time, respectively.

We see from Figures 2.1-2.2 that the robustness of the preconditioned version of the

interior point method, measured by the number of iterations and the number of function evaluations, is slightly worse than that of the unpreconditioned version because of the change in the shape of the trust regions, which we have explained in the previous section. This happens for all the values of `pmem`. However, in terms of efficiency, measured by the average number of CG iterations and CPU time, the performance of the solver is dramatically improved by the incomplete Cholesky preconditioners, as seen from Figures 2.3-2.4. This is due to the fact that the term $A_I^T \Sigma A_I$ is a diagonal matrix for bound constraints, which is cheap to form and does not add to the density of the Hessian of Lagrangian $\nabla^2_{xx}\mathcal{L}$. Thus the incomplete Cholesky factorization of the condensed Hessian $\nabla^2_{xx}\mathcal{L} + A_I^T \Sigma A_I$ can be performed at a reasonably low cost and provides a very good preconditioner.

Moreover, the value of `pmem` also affects the efficiency of the interior method in a meaningful way: The larger value `pmem` takes, the closer is LL^T to the condensed Hessian, the better preconditioner it gives, the less CG iterations are taken and the more CPU time is reduced.

Results for General NLPs with Only Inequality Constraints

Figures 2.5-2.8 show the experiment results on the problems with only inequality constraints. We also observe from Figures 2.5-2.6 that the robustness of the interior point method is slightly deteriorated by the preconditioners, again explained by the change in the shape of the trust regions.

In terms of efficiency, we also notice that the average number of CG iterations, as expected, is dramatically reduced by the preconditioners, as shown by Figure 2.7. Generally speaking, the larger value `pmem` takes, the less CG iterations are taken. In terms of CPU time, Figure 2.8 gives the performance profile based on 37 problems for which all the solvers tested converge and the CPU times are larger than 0.1 second, in order to make a more accurate

comparison. We see that the preconditioners help to reduce the CPU time substantially.

On the other hand, we predict that the unpreconditioned version of the interior method could be faster than the ones with the preconditioners for larger scale problems with dense rows in the constraint Jacobian A_I, because in this case the condensed Hessian will be dense, and its incomplete Cholesky factorization will not be applicable although it may still give good preconditioners. Further numerical experiments are to be conducted to show how useful the incomplete Cholesky preconditioners are for these problems. The incomplete Cholesky preconditioners with dense rows of A_I properly dropped, as proposed in Section 2.4.4, are to be investigated with care.

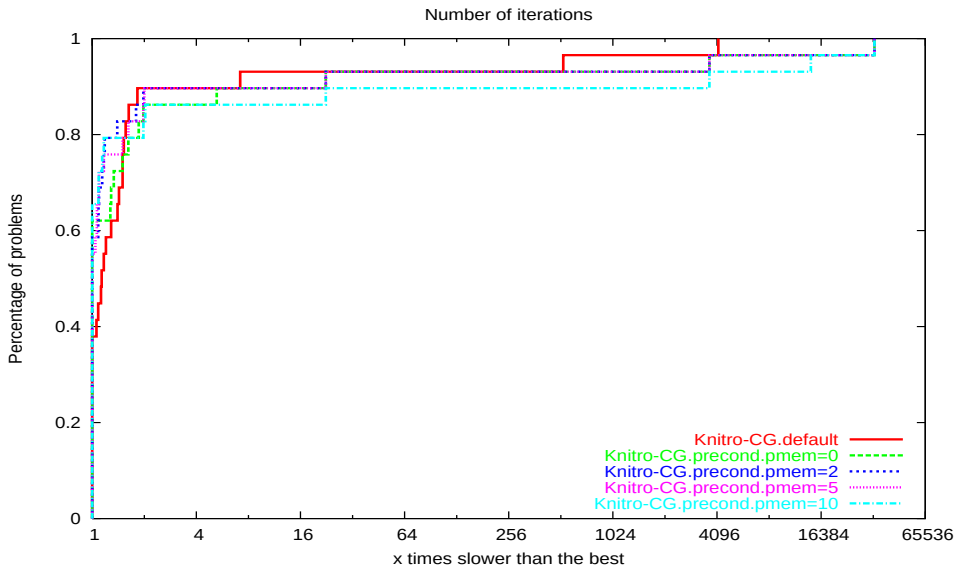


Figure 2.1: Performance profile for KNITRO/CG on 29 bound constrained problems, number of iterations.

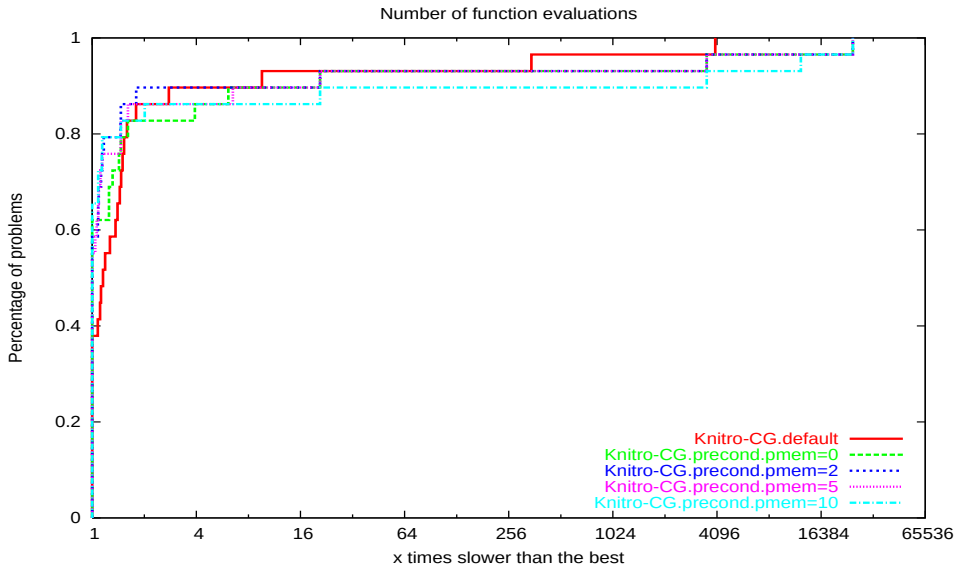


Figure 2.2: Performance profile for KNITRO/CG on 29 bound constrained problems, number of function evaluations.

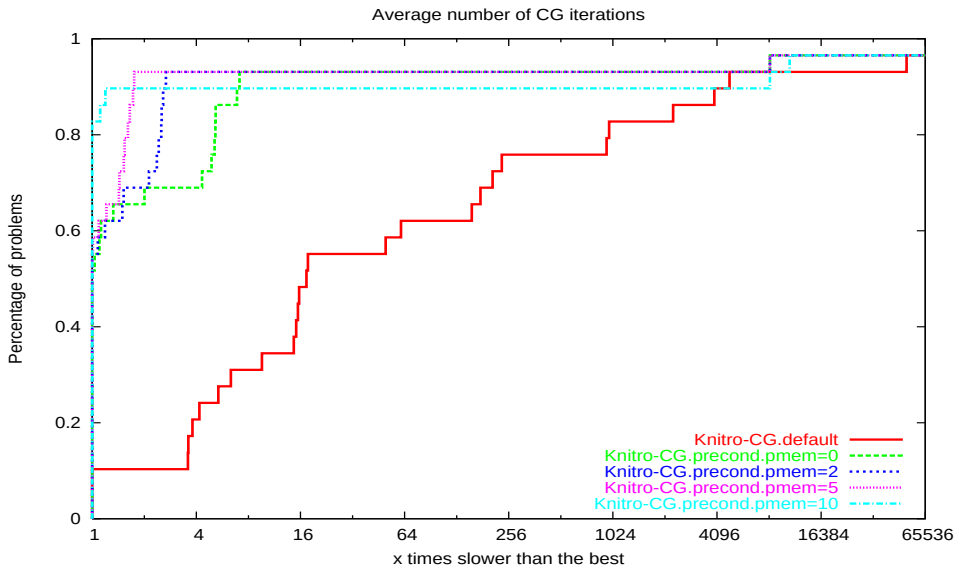


Figure 2.3: Performance profile for KNITRO/CG on 29 bound constrained problems, average number of CG iterations.

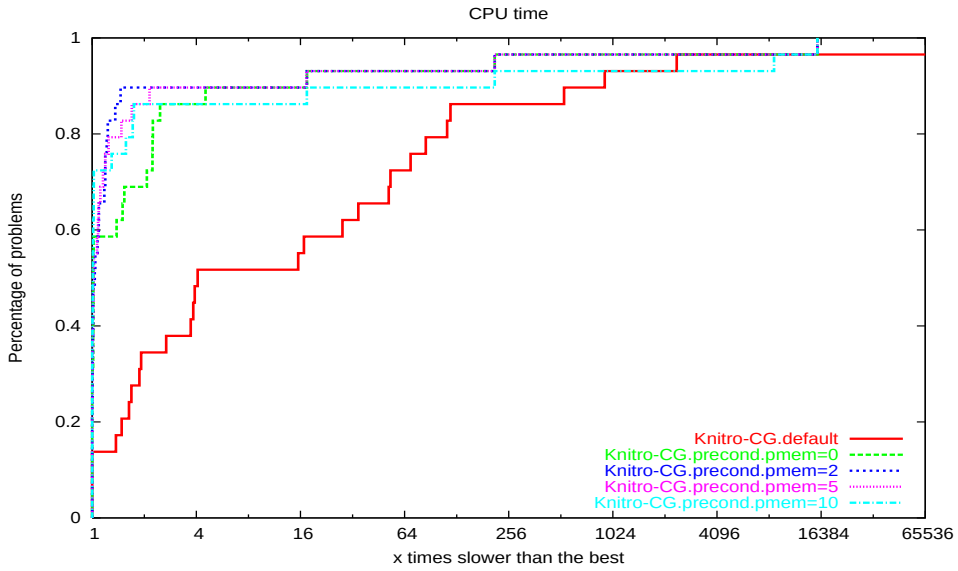


Figure 2.4: Performance profile for KNITRO/CG on 29 bound constrained problems, CPU time.

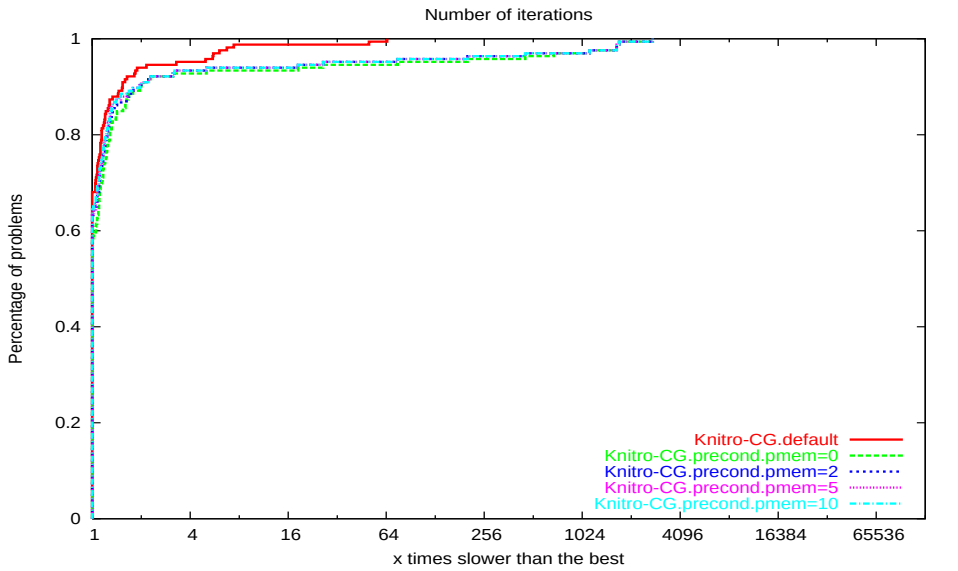


Figure 2.5: Performance profile for KNITRO/CG on problems with only inequality constraints, number of iterations.

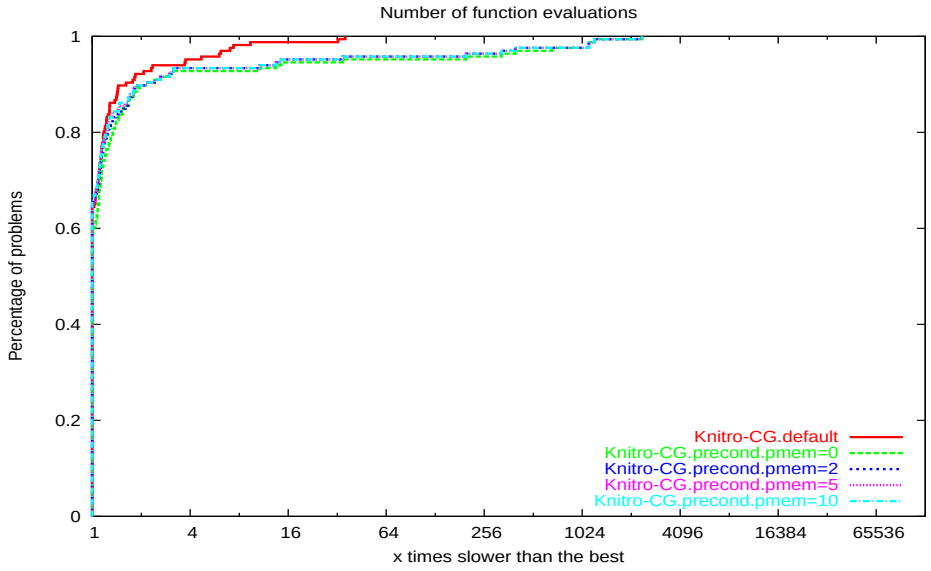


Figure 2.6: Performance profile for KNITRO/CG on problems with only inequality constraints, number of function evaluations.

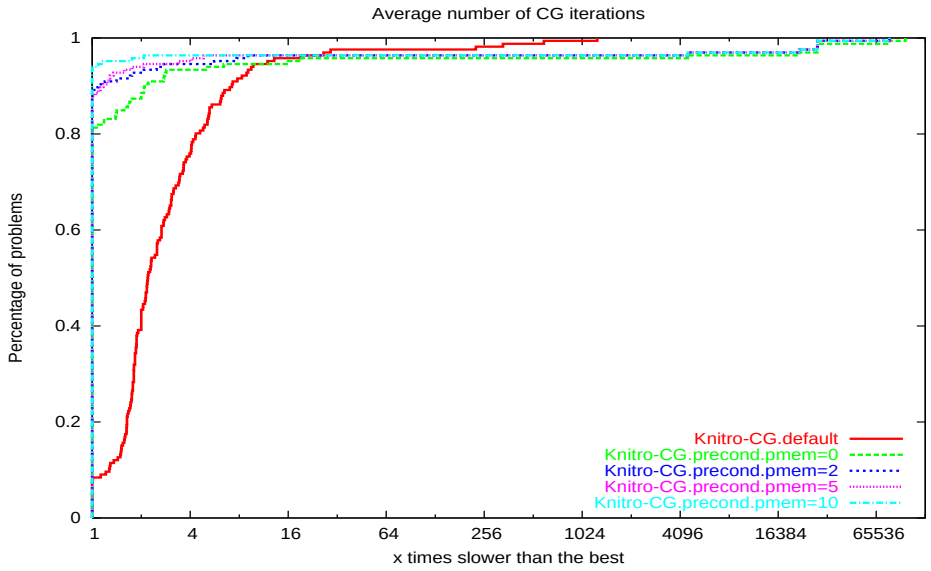


Figure 2.7: Performance profile for KNITRO/CG on problems with only inequality constraints, average number of CG iterations.

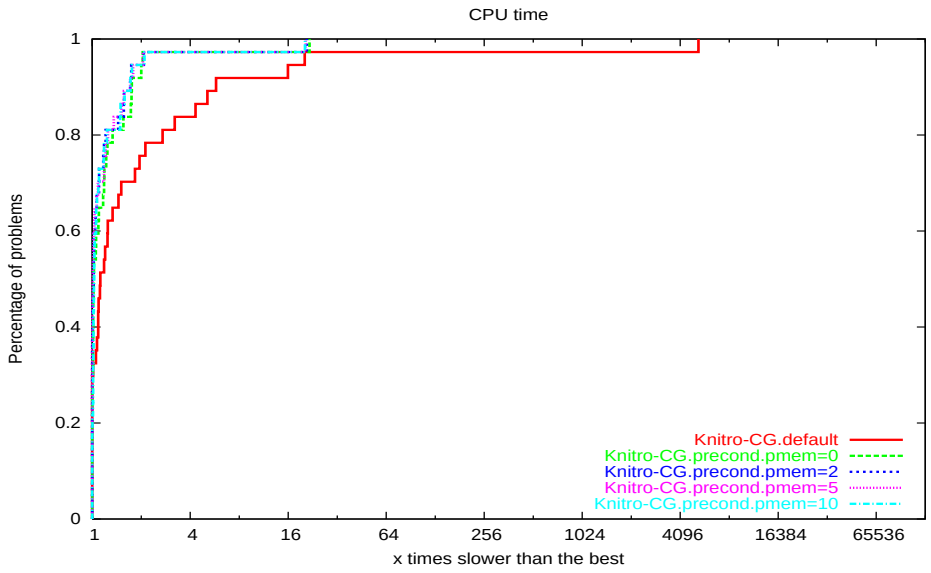


Figure 2.8: Performance profile for KNITRO/CG on problems with only inequality constraints, CPU time (reduced problem set).

Chapter 3

A Strategic Bidding Problem

3.1 Introduction

In this chapter we study a strategic bidding problem in a short term electricity market and consider solving it by nonlinear programming techniques. In this market, different companies that own electricity generation plants compete with each other to provide electricity for the market in a given period of time. On a timely basis, each company has to propose how much to charge for each unit of the electricity production at each individual plant they own, and/or the production capacity of each individual plant. These proposals are called bids, and the plants are called bidders or generators. Note that the bids from the generators belonging to the same company are not necessarily the same.

There is an independent operator who gets all the bids from the companies and determines a production schedule that satisfies the electricity demand of the market. The independent operator solves an economic dispatch problem in order to find the production schedule and a spot price that minimize the aggregated possible cost for the consumers. Once the production schedule and the spot price are determined, each generator in the system is paid the spot price (instead of its own bid) for each unit of its electricity production. Hence the profit of

each company is determined by the difference between the spot price and the production cost of the generators belonging to this company.

The strategic bidding problem is defined for a specific company that owns several generators in the market. The company would like to find the optimal bids that maximize the company's total profit, assuming the bids from other companies are known (we will discuss later how to relax this assumption). The main constraint is that the independent operator determines the production schedule by solving the economic dispatch problem. Therefore this problem can be seen as a bilevel optimization problem. We will provide later in this chapter the details of the problem formulation. See also Hobbs, Metzler and Pang [26], Weber and Overbye [45] and Pereira, Granville, Fampa, Dix and Barroso [36].

As indicated above, the bid of a generator may, in general, include two parts: a price bid and a capacity bid. In our discussion we will assume the production capacity of each plant in the system is a constant and is not part of the decision variables. The optimization model under this assumption is called the Bertrand model (see Fudenberg and Tirole [19]).

3.1.1 The Independent Operator's Problem

In this section we discuss the independent operator's optimization problem and show that it has some characteristics in common with a knapsack problem.

We assume that there are altogether n bids/generators in the system, where only t of them are from Company A ($n > t$). We denote the bids from Company A by $x \in \Re^t$, and the bids outside Company A by $b \in \Re^{n-t}$. Our goal is to solve the strategic bidding problem of Company A.

The independent operator solves the economic dispatch problem for the generation schedule $y \in \Re^n$, given the bids from all the n generators. Given the bids x and b, the economic

dispatch problem is

$$\underset{y}{\text{minimize}} \quad \begin{bmatrix} x \\ b \end{bmatrix}^T y \tag{3.1a}$$

$$\text{subject to} \quad y^T e \geq d \quad (\text{multiplier: } \lambda) \tag{3.1b}$$

$$0 \leq y \leq E \quad (\text{multiplier: } \pi_l, \pi_u), \tag{3.1c}$$

where $e \in \Re^n$ is the vector of all ones, $E \in \Re^n$ is the vector of generators' capacities and d is the demand of the market. In the Bertrand model we assume that E is a known vector. The multiplier λ associated with the demand constraint (3.1b) is called the spot price, as it determines how much the objective function would change if the demand increases by one unit.

Note that the economic dispatch problem (3.1) is a linear program that it is similar to the knapsack problem. We can solve it by making the generator with the lowest bid produce at its full capacity, unless the market demand is satisfied. When there are no equal bids, we know that at the solution of the problem (3.1), there will be some elements of y that are at their upper bounds, some elements of y that are zero, and at most one element of y that is between its upper bound and zero. We also know that the demand constraint (3.1b) is always active at the solution of the economic dispatch problem.

3.1.2 Dual Formulation of the Operator's Problem

The dual problem of (3.1) is

$$\underset{\lambda,\pi_l,\pi_u}{\text{maximize}} \quad d\lambda - E^T\pi_u \tag{3.2a}$$

$$\text{subject to} \quad \lambda e - \pi_u + \pi_l = \begin{bmatrix} x \\ b \end{bmatrix} \tag{3.2b}$$

$$\lambda \geq 0, \pi_l \geq 0, \pi_u \geq 0. \tag{3.2c}$$

Constraint (3.2b) consists of n equations. For the moment we assume that there are no equal bids in the market. In the previous section we have mentioned that at the solution of the economic dispatch problem there is at most one element of the production schedule between its upper and lower bounds. Suppose we know that $0 < y_k < E_k$ holds for a certain $1 \leq k \leq n$. Then it is easy to see that $(\pi_l)_k = (\pi_u)_k = 0$ because the bounds on y_k are both inactive. Hence the k-th equation in (3.2b) shows that

$$\lambda = \left(\begin{bmatrix} x \\ b \end{bmatrix} \right)_k .$$

This indicates that the bid of the k-th generator is the spot price of the market. In other words, when there are no equal bids, if a generator produces electricity for the market, but not at its full capacity, then its bid is the spot price.

From constraint (3.2b) we have

$$\pi_u = \lambda e + \pi_l - \begin{bmatrix} x \\ b \end{bmatrix} .$$

We eliminate π_u from the dual problem (3.2) using the above equation and get

$$\begin{aligned} \underset{\lambda,\pi_l}{\text{maximize}} \quad & (d - E^T e)\lambda - E^T \pi_l + V \\ \text{subject to} \quad & \lambda \geq 0, \pi_l \geq 0, \lambda e + \pi_l - \begin{bmatrix} x \\ b \end{bmatrix} \geq 0, \end{aligned}$$

where $V \stackrel{\text{def}}{=} E^T \begin{bmatrix} x \\ b \end{bmatrix}$. We consider the solution of this problem in three cases:

(i) When $d > E^T e$: The above problem is unbounded above, because we may set $\pi_l = 0$, and the larger value λ takes, the larger objective value we will get. In this case the strong duality property of linear program does not hold. That is, the optimal primal objective does not necessarily equal the optimal dual objective. In our strategic model, if the demand is larger than the total production capacity of the generators, all the generators will produce at full capacity and can bid as large as they want.

(ii) When $d = E^T e$: The above problem becomes trivial and has infinite number of solutions at any value of λ and $\pi_l = 0$. The optimal objective value is V. This indicates that if the system demand is exactly what the generators can provide, all the generators will produce at full capacity and the independent operator can decide the spot price.

(iii) When $d < E^T e$: The above problem has a unique solution and finite optimal objective value. In practice, if the primal problem (3.1) is well defined, we should have $d < E^T e$ because otherwise the demand constraint (3.1b) cannot be satisfied even if every generator is producing at full capacity. In this case the strong duality property holds, which indicates that the optimal primal objective equals the optimal dual objective.

3.1.3 Bilevel Optimization Formulation

We are now ready to present the strategic bidding problem and its bilevel formulation.

Company A would like to determine its optimal bids x such that the total profit by all the generators belonging to this company is maximized. Since each generator is paid the spot price, while the spot price and the production schedule are obtained from the economic dispatch problem by the independent operator after all the bids are collected, the strategic bidding problem for Company A can be formulated as

$$\underset{x,y,\lambda,\pi_l,\pi_u}{\text{maximize}} \quad (\lambda e(1:t) - c_1)^T y(1:t) \tag{3.4a}$$

$$\text{subject to} \quad x \geq c_1 \tag{3.4b}$$

$$y \text{ solves (3.1) with multipliers } \lambda, \pi_l, \pi_u, \tag{3.4c}$$

where $c_1 \in \Re^t$ and $c_2 \in \Re^{n-t}$ are the operating costs of the generators from Company A and from outside Company A, respectively. Constraint (3.4a) makes intuitive sense because each generator would not like to bid lower than its own operating cost.

In practice, the company can only decide the bids x, while the production schedule y and all the multipliers λ, π_l and π_u are not controlled by the company. In the optimization problem, however, the decision variables include the bids of Company A, the production schedule, and the multipliers associated with all the constraints of the economic dispatch problem.

3.2 Nonlinear Programming Formulations of the Problem

One way to solve the strategic bidding problem is to reformulate it into a single level optimization problem. In this section we discuss two different nonlinear programming formulations of the strategic bidding problem (3.4), namely, the MPCC formulation and one based on strong duality. MPCC stands for mathematical program with complementarity constraints.

3.2.1 MPCC Formulation

We have noted that the economic dispatch problem (3.1) is a linear program. Therefore its first order necessary optimality conditions are sufficient to characterize a solution. In other words, any point satisfying these optimality conditions must be a global solution of the problem, but it is not necessarily the unique global solution of (3.1).

As a common approach to reformulate a bilevel optimization problem, the MPCC formulation of the strategic bidding problem replaces the lower level problem in (3.4) by its first order necessary optimality conditions. After reorganizing the terms, we have the MPCC

formulation

$$\underset{x,y,\lambda,\pi_l,\pi_u}{\text{maximize}} \quad (\lambda e(1:t) - c_1)^T y(1:t) \tag{3.5a}$$

$$\text{subject to} \quad x \geq c_1 \tag{3.5b}$$

$$\lambda e - \pi_u + \pi_l = \begin{bmatrix} x \\ b \end{bmatrix} \tag{3.5c}$$

$$0 \leq \lambda \perp \left(y^T e - d\right) \geq 0 \tag{3.5d}$$

$$0 \leq \pi_l \perp y \geq 0 \tag{3.5e}$$

$$0 \leq \pi_u \perp (E - y) \geq 0. \tag{3.5f}$$

Here we use the complementarity notation $\perp$ to indicate some logical conditions. Specifically, $0 \leq \pi_l \perp y \geq 0$ stands for

$$\pi_l \geq 0,\ y \geq 0 \text{ and either } (\pi_l)_i = 0 \text{ or } y_i = 0 \text{ for all } 1 \leq i \leq n.$$

The variables of problem (3.5) are $x \in \Re^t$, $y \in \Re^n$, $\lambda \in \Re^1$, $\pi_l \in \Re^n$ and $\pi_u \in \Re^n$. The total number of variables is $3n + t + 1$. The total number of constraints is $7n + t + 3$, of which the number of equality constraints is $3n + 1$ and the number of inequality constraints is $4n + t + 2$.

Problem (3.5) includes complementarity constraints (3.5d)-(3.5f), and therefore is a Mathematical Program with Complementarity Constraints (MPCC). Since it is widely known that the constraint qualification conditions LICQ and MFCQ do not hold at every feasible point of an MPCC, care needs to be used when (3.5) is solved numerically.

3.2.2 Strong Duality Formulation

Another way to reformulate the bilevel optimization problem (3.4) as a nonlinear program is to use the strong duality condition for linear programming. That is, we require that the lower level linear program is primal feasible and dual feasible, and that the primal objective value equals the dual objective value. With these requirements, the strategic bidding problem for Company A is

$$\underset{x,y,\lambda,\pi_l,\pi_u}{\text{maximize}} \quad (\lambda e(1:t) - c_1)^T y(1:t) \tag{3.6a}$$

$$\text{subject to} \quad x \geq c_1 \tag{3.6b}$$

$$y^T e \geq d \tag{3.6c}$$

$$0 \leq y \leq E \tag{3.6d}$$

$$\lambda e - \pi_u + \pi_l = \begin{bmatrix} x \\ b \end{bmatrix} \tag{3.6e}$$

$$\lambda \geq 0, \pi_l \geq 0, \pi_u \geq 0 \tag{3.6f}$$

$$\begin{bmatrix} x \\ b \end{bmatrix}^T y = d\lambda - E^T \pi_u. \tag{3.6g}$$

The variables of problem (3.6) are $x \in \Re^t$, $y \in \Re^n$, $\lambda \in \Re^1$, $\pi_l \in \Re^n$ and $\pi_u \in \Re^n$. The total number of variables is $3n + t + 1$. The total number of constraints is $5n + t + 3$, of which the number of equality constraints is $n + 1$ and the number of inequality constraints is $4n + t + 2$. More specifically, the problem has a quadratic objective function. Constraints (3.6b)-(3.6f) are all linear, while (3.6g) is the only nonlinear constraint, which is quadratic.

3.2.3 Existing Methods and Heuristics

In this section we will briefly talk about the existing methods and heuristics used to solve the two nonlinear programming formulations of the strategic bidding problem. More detailed discussion can be found in Pereira, Granville, Dix and Barroso [35] and the references therein.

The MPCC formulation (3.5) of the strategic bidding problem has complementarity constraints, so special techniques must be used when it is solved numerically. KNITRO (Byrd, Nocedal and Waltz [8]) is able to solve MPCCs using the interior-penalty algorithm proposed in Leyffer, López-Calva and Nocedal [29]. The active set SQP method FILTER (Fletcher and Leyffer [16, 15]) is also able to handle this type of problems. An exact penalty approach is proposed by White and Anandalingam [46], where the complementarity constraints are penalized.

The strong duality formulation (3.6) has a quadratic objective function and linear and quadratic constraints. Two types of methods have been used to solve this formulation.

The first is to use a general nonlinear programming solver. The second type of method is to penalize the quadratic equality constraint (3.6g) using an l_1 approach and solve the

penalization problem

$$\underset{x,y,\lambda,\pi_l,\pi_u}{\text{maximize}} \quad (\lambda e(1:t) - c_1)^T y(1:t) - \mu \left(\begin{bmatrix} x \\ b \end{bmatrix}^T y - d\lambda + E^T \pi_u \right) \tag{3.7a}$$

$$\text{subject to} \quad x \geq c_1 \tag{3.7b}$$

$$y^T e \geq d \tag{3.7c}$$

$$0 \leq y \leq E \tag{3.7d}$$

$$\lambda e - \pi_u + \pi_l = \begin{bmatrix} x \\ b \end{bmatrix} \tag{3.7e}$$

$$\lambda \geq 0, \pi_l \geq 0, \pi_u \geq 0. \tag{3.7f}$$

Note that when the economic dispatch problem (3.1) and its dual problem (3.2) are both feasible, the weak duality condition of linear programming gives

$$\begin{bmatrix} x \\ b \end{bmatrix}^T y \geq d\lambda - E^T \pi_u,$$

hence the absolute value operation is not needed in (3.7a). The objective function of the penalization problem (3.7) has been shown to be an exact penalty function (see Anandalingam and White [1]), thus the solution of the penalization problem is a local solution or stationary point of the original problem (3.6), when the penalty parameter is large enough. The penalization can either be directly solved by a quadratic programming solver, or be approximately solved by a series of linear programs, where the production schedule y is fixed and the resulting linear program is solved for other variables, then these variables are fixed and the resulting linear program in terms of y is solved. This process is repeated until the penalty

gap is small enough.

In order to avoid local optimal solution, some heuristics are proposed when solving the strong duality formulation, such as using different starting points (KNITRO has a `multistart` feature), or performing local searches after a local solution is obtained.

In this thesis we will focus on solving the strong duality formulation of the strategic bidding problem directly, instead of the penalization method.

3.3 Bilevel Program with Linear Lower Level Problem

The strategic bidding problem (3.4) that we have discussed in previous sections is a specific example of a general bilevel optimization problem. More general discussions about bilevel programming can be found in Bard [2] and Dempe [11].

In this section we will study bilevel programs whose lower level problems are linear. Obviously the strategic bidding problem is in this class.

The general form of this class of problems is:

$$\underset{x\in\Re^n, y\in\Re^l}{\text{minimize}} \quad f(x,y) \tag{3.8a}$$

$$\text{subject to} \quad x \text{ solves LP (3.9) with } y \text{ being given data,} \tag{3.8b}$$

where the lower level linear program has the form

$$\underset{x\in\Re^n}{\text{minimize}} \quad c^T x + d^T y \tag{3.9a}$$

$$\text{subject to} \quad Ax + By \geq r \quad (\text{multiplier: } \lambda) \tag{3.9b}$$

$$x \geq 0 \quad (\text{multiplier: } \pi), \tag{3.9c}$$

where we assume that $A \in \Re^{m\times n}$, $B \in \Re^{m\times l}$, $r \in \Re^m$, $c \in \Re^n$, $d \in \Re^l$, $\lambda \in \Re^m$ and $\pi \in \Re^n$.

We note that there is a little difference between the general form (3.8) and the strategic bidding problem. First, the strategic bidding problem has some "outer" level constraints like (3.4b), namely, these constraints are part of the constraints of the bilevel program but not part of the lower level linear program. We will see later in this section that these constraints do not interfere with our analytical results. Second, in problem (3.9), the given vector y appears as the right hand side vector of inequality (3.9b) and serves as parameters of the lower level problem, and the coefficient vector c and d in the lower level objective function are given constants. In the strategic bidding problem, the bids x appear as part of the coefficients of the lower level objective. In this section we will analyze the properties of the constraints of the general form (3.8), and the strategic bidding problem can be analyzed by similar techniques.

Moreover, we also note that the term $d^T y$ is a constant in the objective function of (3.9) with any given value of y. We will still include it in (3.9a) because y is part of the variables of the bilevel problem.

3.3.1 Relation between Two NLP Reformulations

In this section we will discuss the relation between the two nonlinear programming reformulations of a bilevel program with a linear lower level problem. We start from the following low dimensional example and then consider the general case.

Example 3.1 *(An LP as the lower level problem of a bilevel program)*

Consider linear program

$$\underset{x_1,x_2}{\text{minimize}} \quad c_1x_1 + c_2x_2 \tag{3.10a}$$

$$\text{subject to} \quad x_1 + x_2 \geq d \quad (\text{multiplier: } \lambda_1) \tag{3.10b}$$

$$x_1 \geq 0 \quad (\text{multiplier: } \lambda_2). \tag{3.10c}$$

Note that problem (3.10) has a unique finite solution only if $c_1/c_2 > 1$; we assume that this inequality holds. The dual problem of (3.10) is

$$\underset{\lambda_1,\lambda_2}{\text{maximize}} \quad \lambda_1 d \tag{3.11a}$$

$$\text{subject to} \quad c_1 - \lambda_1 - \lambda_2 = 0 \tag{3.11b}$$

$$c_2 - \lambda_1 = 0 \tag{3.11c}$$

$$\lambda_1, \lambda_2 \geq 0. \tag{3.11d}$$

The KKT conditions for (3.10) are:

$$c_1 - \lambda_1 - \lambda_2 = 0 \tag{3.12a}$$

$$c_2 - \lambda_1 = 0 \tag{3.12b}$$

$$x_1 + x_2 - d \geq 0 \tag{3.12c}$$

$$x_1 \geq 0 \tag{3.12d}$$

$$\lambda_1(x_1 + x_2 - d) = 0 \tag{3.12e}$$

$$\lambda_2 x_1 = 0 \tag{3.12f}$$

$$\lambda_1, \lambda_2 \geq 0. \tag{3.12g}$$

Now suppose that (3.10) is the lower level problem of a bilevel program; then (3.12) will be the constraints in the MPCC reformulation of the bilevel program. We add up the complementarity conditions (3.12e)-(3.12f) and group the terms to get

$$x_1(\lambda_1 + \lambda_2) + x_2\lambda_1 = \lambda_1 d.$$

Further making use of the dual feasibility conditions (3.12a)-(3.12b), we get

$$c_1x_1 + c_2x_2 = \lambda_1 d.$$

Note that this is the strong duality property in linear programming, that is, the primal objective equals the dual objective. We then use this equation to replace the complementarity conditions (3.12e)-(3.12f). System (3.12) becomes

$$c_1 - \lambda_1 - \lambda_2 = 0 \tag{3.13a}$$

$$c_2 - \lambda_1 = 0 \tag{3.13b}$$

$$x_1 + x_2 - d \geq 0 \tag{3.13c}$$

$$x_1 \geq 0 \tag{3.13d}$$

$$c_1x_1 + c_2x_2 - \lambda_1 d = 0 \tag{3.13e}$$

$$\lambda_1 \geq 0 \tag{3.13f}$$

$$\lambda_2 \geq 0. \tag{3.13g}$$

It is easy to see that (3.13) are the constraints in the strong duality reformulation of the bilevel program. □

The example above discovers the relation between the MPCC formulation and the strong duality formulation of the bilevel program with linear lower level problem (3.10): We can add

up the complementarity constraints in the MPCC formulation and derive the strong duality formulation using the dual feasibility conditions. We note that the derivation procedure has nothing to do with the objective function of the bilevel program. In fact, Example 3.1 shows that the KKT conditions of linear program (3.10) can be transformed into its strong duality conditions. This inspires us to study the more general linear program of the form (3.9).

We can transform the KKT conditions of linear program (3.9) into its strong duality conditions as follows: First, the KKT conditions of problem (3.9) are

$$c - A^T\lambda - \pi = 0 \tag{3.14a}$$

$$Ax + By - r \geq 0 \tag{3.14b}$$

$$x \geq 0 \tag{3.14c}$$

$$\lambda^T(Ax + By - r) = 0 \tag{3.14d}$$

$$\pi^T x = 0 \tag{3.14e}$$

$$\lambda \geq 0 \tag{3.14f}$$

$$\pi \geq 0. \tag{3.14g}$$

Then we sum up the complementarity conditions (3.14d)-(3.14e) and group terms to get

$$(A^T\lambda + \pi)^T x = (r - By)^T\lambda.$$

Using this equation and (3.14a), we have

$$c^T x = (r - By)^T\lambda.$$

This is the strong duality property of linear program (3.9): the primal objective equals the dual objective (in both of which the constant $d^T y$ is omitted). Hence the strong duality

conditions of (3.9) are obtained by replacing the complementarity conditions (3.14d)-(3.14e) with the strong duality condition in (3.14):

$$c - A^T\lambda - \pi = 0 \tag{3.15a}$$

$$Ax + By - r \geq 0 \tag{3.15b}$$

$$x \geq 0 \tag{3.15c}$$

$$c^T x + (By - r)^T\lambda = 0 \tag{3.15d}$$

$$\lambda \geq 0 \tag{3.15e}$$

$$\pi \geq 0. \tag{3.15f}$$

In the context of bilevel program with linear lower level problem, the above procedure reveals the relation between two nonlinear programming formulations of problem (3.8): Since (3.14) are the constraints of the MPCC formulation and (3.15) are the constraints of the strong duality formulation, we can derive the strong duality formulation of bilevel program (3.8) from its MPCC formulation.

3.3.2 Failure of Constraint Qualifications

We have studied two nonlinear programming reformulations of a bilevel program with a linear lower level problem. It is well known that the constraint qualification conditions LICQ (Definition 1.2) and MFCQ (Definition 1.3), which we describe in Section 1.3.2, fail at any feasible point of an MPCC, which makes MPCCs hard. See Scheel and Scholtes [38] for more details. Therefore the MPCC formulation of a bilevel problem is also hard because of the existence of the complementarity constraints.

Let us now consider the constraint qualifications for the strong duality formulation. Again we will start from the low dimensional example in the previous section by way of motivation,

and then consider the more general case.

Example 3.2 *(Continued with Example 3.1)*

Let us consider the strong duality formulation (3.13) and assume that d is part of the variables of the bilevel program. With respect to variables $(x_1, x_2, d, \lambda_1, \lambda_2)$, the gradients of all the constraints in system (3.13) are:

$$\begin{aligned} v_1 &= (0,0,0,-1,-1)^T \\ v_2 &= (0,0,0,-1,0)^T \\ v_3 &= (1,1,-1,0,0)^T \\ v_4 &= (1,0,0,0,0)^T \\ v_5 &= (c_1,c_2,-\lambda_1,-d,0)^T \\ v_6 &= (0,0,0,1,0)^T \\ v_7 &= (0,0,0,0,1)^T. \end{aligned}$$

Note that constraints (3.13a), (3.13b) and (3.13e) must be active at any feasible point because they are all equality constraints. Equations (3.13a)-(3.13b) give

$$\lambda_1 = c_2 \geq 0, \quad \lambda_2 = c_1 - c_2 \geq 0. \tag{3.16}$$

We first show that at least one of (3.13c) and (3.13f) must be active, and at least one of (3.13d) and (3.13g) must be active. That is, we want to show that

$$\lambda_1(x_1 + x_2 - d) = \lambda_2 x_1 = 0. \tag{3.17}$$

Considering (3.16), we want to show that

$$c_2(x_1 + x_2 - d) = (c_1 - c_2)x_1 = 0.$$

If this is not true, we have from (3.13c)-(3.13g) and (3.16) that $c_2(x_1 + x_2 - d) > 0$, or $(c_1 - c_2)x_1 > 0$. In either of these two cases we know that

$$c_2(x_1 + x_2 - d) + (c_1 - c_2)x_1 > 0.$$

We simplify this inequality and get

$$0 < c_1x_1 + c_2x_2 - c_2d = c_1x_1 + c_2x_2 - \lambda_1 d.$$

This clearly contradicts (3.13e). Hence we know that (3.17) holds. In other words, at least one of (3.13c) and (3.13f) must be active, and at least one of (3.13d) and (3.13g) must be active.

Now we check the constraint qualifications. Recalling that (3.13a), (3.13b) and (3.13e) are always active and their gradients are linearly independent, we consider the following four cases at any feasible point of (3.13):

(i) When (3.13c) and (3.13d) are both active: The gradients of active constraints are v_1, v_2, v_3, v_4 and v_5. Assume that there exists vector $w \in \Re^5$ such that $w^T v_1 = w^T v_2 = w^T v_5 = 0$ and $w^T v_3 > 0$, $w^T v_4 > 0$. From $w^T v_1 = w^T v_2 = 0$ we have $w_4 = w_5 = 0$. From $w^T v_5 = 0$ we have $c_1 w_1 + c_2 w_2 - \lambda_1 w_3 - d w_4 = 0$, hence $c_1 w_1 + c_2 w_2 - \lambda_1 w_3 = 0$. Since $c_2 = \lambda_1 \geq 0$, we have

$$c_1 w_1 + c_2 w_2 - c_2 w_3 = 0. \tag{3.18}$$

On the other hand, from $w^T v_3 > 0$ we have $w_1 + w_2 - w_3 > 0$, hence

$$c_2 w_1 + c_2 w_2 - c_2 w_3 > 0 \tag{3.19}$$

because $c_2 \geq 0$. We then conclude from (3.18)-(3.19) that $(c_1 - c_2) w_1 < 0$. This cannot be true because we know from (3.16) that $c_1 - c_2 = \lambda_2 \geq 0$ and from $w^T v_4 > 0$ that $w_1 > 0$. So such vector w does not exist.

(ii) When (3.13c) and (3.13d) are both inactive: Then (3.17) indicates (3.13f) and (3.13g) are active. The gradients of active constraints are v_1, v_2, v_5, v_6 and v_7. Assume that there exists vector $w \in \Re^5$ such that $w^T v_1 = w^T v_2 = w^T v_5 = 0$ and $w^T v_6 > 0$, $w^T v_7 > 0$. From $w^T v_2 = 0$ we have $w_4 = 0$. From $w^T v_6 > 0$ we have $w_4 > 0$. This contradiction indicates that such vector w does not exist.

(iii) When (3.13c) is active and (3.13d) is inactive: Then (3.17) indicates (3.13g) is active. The gradients of active constraints are v_1, v_2, v_3, v_5 and v_7. Assume that there exists vector $w \in \Re^5$ such that $w^T v_1 = w^T v_2 = w^T v_5 = 0$ and $w^T v_3 > 0$, $w^T v_7 > 0$. From $w^T v_1 = w^T v_2 = 0$ we have $w_4 = w_5 = 0$. From $w^T v_7 > 0$ we have $w_5 > 0$. This contradiction indicates that such vector w does not exist.

(iv) When (3.13d) is active and (3.13c) is inactive: Then (3.17) indicates (3.13f) is active. The gradients of active constraints are v_1, v_2, v_4, v_5 and v_6. Assume that there exists vector $w \in \Re^5$ such that $w^T v_1 = w^T v_2 = w^T v_5 = 0$ and $w^T v_4 > 0$, $w^T v_6 > 0$. From $w^T v_2 = 0$ we have $w_4 = 0$. From $w^T v_6 > 0$ we have $w_4 > 0$. This clear contradiction indicates that such vector w does not exist.

Hence we have shown that MFCQ also fails at any feasible point of (3.13), and then LICQ also fails at any feasible point of (3.13). This is to say, LICQ and MFCQ fail at any feasible

point of the strong duality formulation of the bilevel program whose lower level linear problem is (3.10). □

In order to analyze the constraint qualifications for the general bilevel program (3.8), we eliminate π from system (3.15) and get

$$Ax + By - r \geq 0 \tag{3.20a}$$

$$x \geq 0 \tag{3.20b}$$

$$c^T x + (By - r)^T \lambda = 0 \tag{3.20c}$$

$$\lambda \geq 0 \tag{3.20d}$$

$$c - A^T \lambda \geq 0. \tag{3.20e}$$

We have shown that the complementarity conditions in (3.14) imply the strong duality condition (3.20c). The following lemma shows that the strong duality condition (3.20c), on the other hand, implies the complementarity conditions in (3.14) at a point where the primal and dual linear programs are both feasible. Therefore we know that the complementarity conditions are equivalent to the strong duality condition for a linear program.

Lemma 3.3 *At any feasible point of system* (3.20)*, the i-th constraint of* (3.20a) *must be complementary to the i-th constraint of* (3.20d)*, and the j-th constraint of* (3.20b) *must be complementary to the j-th constraint of* (3.20e)*. That is,*

$$(Ax + By - r)_i \lambda_i = 0 \quad (\forall 1 \leq i \leq m), \quad x_j (c - A^T \lambda)_j = 0 \quad (\forall 1 \leq j \leq n). \tag{3.21}$$

Proof. We use contradiction to prove the lemma.

At any feasible point of (3.20), assume that there exists an index $1 \leq i \leq m$ such that the i-th constraint of (3.20a) is not complementary to the i-th constraint of (3.20d). In other

words, $(Ax + By - r)_i > 0$ and $\lambda_i > 0$. Then we have from (3.20) that

$$\begin{aligned} c^T x = (r - By)^T \lambda &= (r - By)_i \lambda_i + \sum_{1 \le k \le m, k \ne i} (r - By)_k \lambda_k \\ &< (Ax)_i \lambda_i + \sum_{1 \le k \le m, k \ne i} (Ax)_k \lambda_k = \lambda^T A x = (A^T \lambda)^T x. \end{aligned}$$

Hence we have $(c - A^T \lambda)^T x < 0$, which contradicts the fact that at any feasible point (3.20b) and (3.20e) hold. This indicates that $(Ax + By - r)_i \lambda_i = 0$ for all $1 \le i \le m$.

The second part of (3.21) can be shown similarly: Assume that there exists an index $1 \le j \le n$ such that the j-th constraint of (3.20b) is not complementary to the j-th constraint of (3.20e). In other words, $x_j > 0$ and $(c - A^T \lambda)_j > 0$. We have

$$\begin{aligned} (r - By)^T \lambda = c^T x &= c_j x_j + \sum_{1 \le k \le n, k \ne j} c_k x_k \\ &> (A^T \lambda)_j x_j + \sum_{1 \le k \le n, k \ne j} (A^T \lambda)_k x_k = x^T A^T \lambda = (Ax)^T \lambda. \end{aligned}$$

Hence $(r - By - Ax)^T \lambda > 0$, which contradicts the fact that at any feasible point (3.20a) and (3.20d) hold. This indicates that $x_j (c - A^T \lambda)_j = 0$ for all $1 \le j \le n$. The proof is completed. □

The following theorem is our main conclusion:

Theorem 3.4 *LICQ and MFCQ both fail at every feasible point of the strong duality formulation of bilevel program* (3.8)*, with* (3.9) *as the lower level problem.*

Proof. It is sufficient to show that LICQ and MFCQ both fail at every feasible point of nonlinear system (3.20).

Let us consider MFCQ. According to Lemma 3.3 and without loss of generality, we assume

that there exist index sets $J_1 \in \{1, 2, \cdots, m\}$, $I_1 \in \{1, 2, \cdots, n\}$ such that

$$(Ax + By - r)^{(J_1)} = 0 \qquad \lambda^{(J_1)} \geq 0 \tag{3.22a}$$

$$(Ax + By - r)^{(J_2)} > 0 \qquad \lambda^{(J_2)} = 0 \tag{3.22b}$$

$$x^{(I_1)} = 0 \qquad (c - A^T\lambda)^{(I_1)} \geq 0 \tag{3.22c}$$

$$x^{(I_2)} > 0 \qquad (c - A^T\lambda)^{(I_2)} = 0, \tag{3.22d}$$

where $J_2 = \{1, 2, \cdots, m\} \backslash J_1$ and $I_2 = \{1, 2, \cdots, n\} \backslash I_1$. Note that the following derivation also holds when any index set J_1, J_2, I_1 or I_2 is empty.

Assume there exists a vector $d = \begin{bmatrix} d_x \\ d_y \\ d_\lambda \end{bmatrix} \in \Re^{n+l+m}$ such that

$$c^T d_x + \lambda^T B d_y + (By - r)^T d_\lambda = 0 \tag{3.23a}$$

$$(Ad_x)^{(J_1)} + (Bd_y)^{(J_1)} > 0 \tag{3.23b}$$

$$d_x^{(I_1)} > 0 \tag{3.23c}$$

$$d_\lambda^{(J_2)} > 0 \tag{3.23d}$$

$$-(A^T d_\lambda)^{(I_2)} > 0. \tag{3.23e}$$

Therefore by (3.22c), (3.22d) and (3.23c),

$$(c^{(I_1)})^T d_x^{(I_1)} \geq \left((A^T\lambda)^{(I_1)}\right)^T d_x^{(I_1)} = \lambda^T A^{(I_1)} d_x^{(I_1)} \tag{3.24a}$$

$$(c^{(I_2)})^T d_x^{(I_2)} = \left((A^T\lambda)^{(I_2)}\right)^T d_x^{(I_2)} = \lambda^T A^{(I_2)} d_x^{(I_2)}. \tag{3.24b}$$

Also noticing that $\lambda^{(J_2)} = 0$, we have

$$
\begin{aligned}
& c^T d_x + \lambda^T B d_y \\
= \; & c^T d_x + \begin{bmatrix} \lambda^{(J_1)} \\ \lambda^{(J_2)} \end{bmatrix}^T \begin{bmatrix} (Bd_y)^{(J_1)} \\ (Bd_y)^{(J_2)} \end{bmatrix} \\
= \; & c^T d_x + \left(\lambda^{(J_1)}\right)^T (Bd_y)^{(J_1)} + \left(\lambda^{(J_2)}\right)^T (Bd_y)^{(J_2)} \\
= \; & (c^{(I_1)})^T d_x^{(I_1)} + (c^{(I_2)})^T d_x^{(I_2)} + \left(\lambda^{(J_1)}\right)^T (Bd_y)^{(J_1)} + 0 \\
\geq \; & \lambda^T A^{(I_1)} d_x^{(I_1)} + \lambda^T A^{(I_2)} d_x^{(I_2)} + \left(\lambda^{(J_1)}\right)^T (Bd_y)^{(J_1)} \quad \text{(by (3.24))} \\
= \; & \lambda^T \left[A^{(I_1)} \; A^{(I_2)}\right] \begin{bmatrix} d_x^{(I_1)} \\ d_x^{(I_2)} \end{bmatrix} + \left(\lambda^{(J_1)}\right)^T (Bd_y)^{(J_1)} \\
= \; & \lambda^T A d_x + \left(\lambda^{(J_1)}\right)^T (Bd_y)^{(J_1)} \\
= \; & \begin{bmatrix} \lambda^{(J_1)} \\ \lambda^{(J_2)} \end{bmatrix}^T \begin{bmatrix} A^{(J_1)} d_x \\ A^{(J_2)} d_x \end{bmatrix} + \left(\lambda^{(J_1)}\right)^T (Bd_y)^{(J_1)} \\
= \; & \left(\lambda^{(J_1)}\right)^T \left(A^{(J_1)} d_x + (Bd_y)^{(J_1)}\right) + \left(\lambda^{(J_2)}\right)^T \left(A^{(J_2)} d_x\right) \\
\geq \; & 0^T 0 + 0^T \left(A^{(J_2)} d_x\right) \quad \text{(by (3.22a), (3.22b) and (3.23b))} \\
= \; & 0. \qquad (3.25)
\end{aligned}
$$

Similarly, by (3.22a), (3.22b) and (3.23d),

$$\left((By - r)^{(J_1)}\right)^T d_\lambda^{(J_1)} = -\left((Ax)^{(J_1)}\right)^T d_\lambda^{(J_1)} = -x^T \left(A^{(J_1)}\right)^T d_\lambda^{(J_1)} \qquad (3.26a)$$

$$\left((By - r)^{(J_2)}\right)^T d_\lambda^{(J_2)} > -\left((Ax)^{(J_2)}\right)^T d_\lambda^{(J_2)} = -x^T \left(A^{(J_2)}\right)^T d_\lambda^{(J_2)}. \qquad (3.26b)$$

So we have

$$\begin{aligned}
& (By-r)^T d_\lambda \\
= \; & \left((By-r)^{(J_1)}\right)^T d_\lambda^{(J_1)} + \left((By-r)^{(J_2)}\right)^T d_\lambda^{(J_2)} \\
> \; & -x^T \left(A^{(J_1)}\right)^T d_\lambda^{(J_1)} - x^T \left(A^{(J_2)}\right)^T d_\lambda^{(J_2)} \quad \text{(by (3.26))} \\
= \; & -x^T \left(\begin{bmatrix} A^{(J_1)} \\ A^{(J_2)} \end{bmatrix} \right)^T \begin{bmatrix} d_\lambda^{(J_1)} \\ d_\lambda^{(J_2)} \end{bmatrix} \\
= \; & -x^T A^T d_\lambda \\
= \; & -\left(\begin{bmatrix} x^{(I_1)} \\ x^{(I_2)} \end{bmatrix} \right)^T \begin{bmatrix} \left(A^{(I_1)}\right)^T \\ \left(A^{(I_2)}\right)^T \end{bmatrix} d_\lambda \\
= \; & -\left(x^{(I_1)}\right)^T \left(A^{(I_1)}\right)^T d_\lambda - \left(x^{(I_2)}\right)^T \left(A^{(I_2)}\right)^T d_\lambda \\
> \; & -0^T \left(A^{(I_1)}\right)^T d_\lambda + 0^T 0 \quad \text{(by (3.22c), (3.22d) and (3.23e))} \\
= \; & 0. \qquad (3.27)
\end{aligned}$$

Thus (3.25) and (3.27) give

$$c^T d_x + \lambda^T B d_y + (By-r)^T d_\lambda > 0,$$

which contradicts (3.23a). So such vector d does not exist.

Hence we know that MFCQ fails at every feasible point of (3.20), then LICQ also fails at every feasible point of (3.20). Therefore we conclude that LICQ and MFCQ both fail at every feasible point of the strong duality formulation of bilevel program (3.8). □

For a bilevel program with linear lower level problem (3.9), Theorem 3.4 indicates that the strong duality formulation will be as difficult for any general nonlinear programming solver as the MPCC formulation.

In the beginning of this section, we have mentioned that the strategic bidding problem has other "outer" level constraints that are not included in the general form (3.8) of the bilevel program that we have analyzed. We point out that these constraints are all lower bound constraints, thus the existence of these constraints will not reduce the number of active constraints at any feasible point. Instead, they may only provide the strong duality formulation with more gradient vectors of the form $(\epsilon_i^T, 0, 0)^T$, where ϵ_i indicates a coordinate vector, that are the same as those of the constraints (3.9c). Therefore one can easily show that our analysis in this section still holds well when these extra bound constraints are present.

3.4 Discontinuous Objective Function of Bilevel Programs

In the previous section we have analyzed the properties of the constraints of the strong duality reformulation of a bilevel program with a linear lower level problem. The failure of the constraint qualifications may or may not be a source of difficulties for the strong duality formulation. In this section we analyze an important property of the objective function value of bilevel programs, namely, the objective function may not be continuous. This property makes the strong duality formulation of the bilevel program hard for nonlinear optimization solvers.

3.4.1 Objective of the Strategic Bidding Problem

We first consider the objective function of the bilevel formulation of the strategic bidding problem (3.4).

For simplicity we assume that the production cost $c = 0$. Then the net profit of Company

A, the objective of the strategic bidding problem, is given by $\lambda e(1:t)^T y(1:t)$. Let us consider the example where $t = 1$ and $E^T e > d$, which means that there is only one generator belonging to Company A, and the total production capacity of the market is larger than the system demand. Without loss of generality, we rank the bids from outside Company A to be $b_1 \leq b_2 \leq \cdots \leq b_{n-1}$. Let k be the minimum index such that $E_A + \sum_{i=1}^{k} E_{B_k} > d$, where E_A is the production capacity of the (only) generator belonging to Company A, and E_{B_i} is the production capacity of the i-th generator outside Company A, whose bid is b_i.

When $t = 1$, the lower level linear program can be solved by a supply-demand graph because of its knapsack-like property. The relationship between the bid x and the production level y_A of the generator belonging to Company A is the step-like function shown in Figure 3.1. Obviously for the lower problem the production level of Company A is not a continuous function of the bid x.

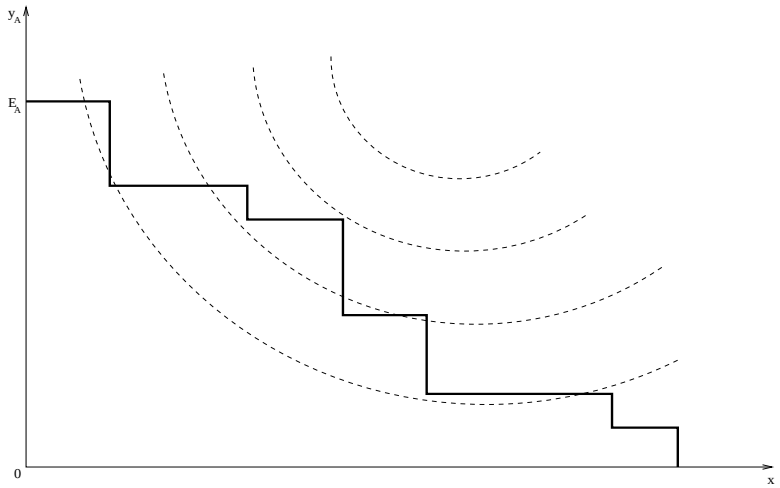


Figure 3.1: The objective value of the strategic bidding problem projected onto the x-y_A plane.

Now consider the full strategic bidding problem (3.4). Figure 3.2 shows the dependence of the profit of Company A on the bid x. We consider the following cases:

- When the bid x is less than b_k, the spot price of the system is determined by b_k, and

the generator belonging to Company A is always producing at its full capacity, so its profit is a constant $b_k E_A$. When x is equal to b_k, the generator belonging to Company A is making a bid that is equal to some other generators outside Company A, so the independent operator may choose any combination of the production schedule in order to satisfy the demand, and thus the profit of Company A is not uniquely determined.

- When the bid x exceeds that of some outside generator b_{k+j} where $j \geq 1$ but is less than b_{k+j+1}, x determines the spot price (i.e. $\lambda = x$), but the generator belonging to Company A no longer produces at its full capacity. Instead, the amount of its production is $y_A = d - \sum_{1 \leq i \leq j} E_{B_i}$. Hence the profit of Company A is $\lambda y_A = \left(d - \sum_{1 \leq i \leq j} E_{B_i}\right) x$, which is a linear function of x, and the slope of this linear function is y_A. It can be seen that the value of y_A is decreasing as x increases, so the slope of the linear segments in Figure 3.2 is decreasing. Again, when the bid x is equal to some other bid, the profit of Company A is not uniquely determined because the independent operator can choose any combination of the production schedule to meet the demand.

- When the bid x is greater than a certain value b_{k+l}, the generator belonging to Company A will not be included in the production schedule, that is, $y_A = 0$. In this case the profit of Company A is 0.

Figure 3.2 shows that the objective function of the strategic bidding problem is generally discontinuous. Moreover, the problem has many local minima. Therefore, the strategic bidding problem is numerically hard for any solver.

In Figure 3.1, we imagine that the third dimension is the profit $z = \lambda y_A$. If projected on the x-y_A plane, the profit seems to be continuous and differentiable, as shown by the dashed curves in Figure 3.1. However, we can see if the profit is restricted to the feasible (x, y_A) pairs on the piecewise linear segments, it is no longer continuous. Rather, the value of

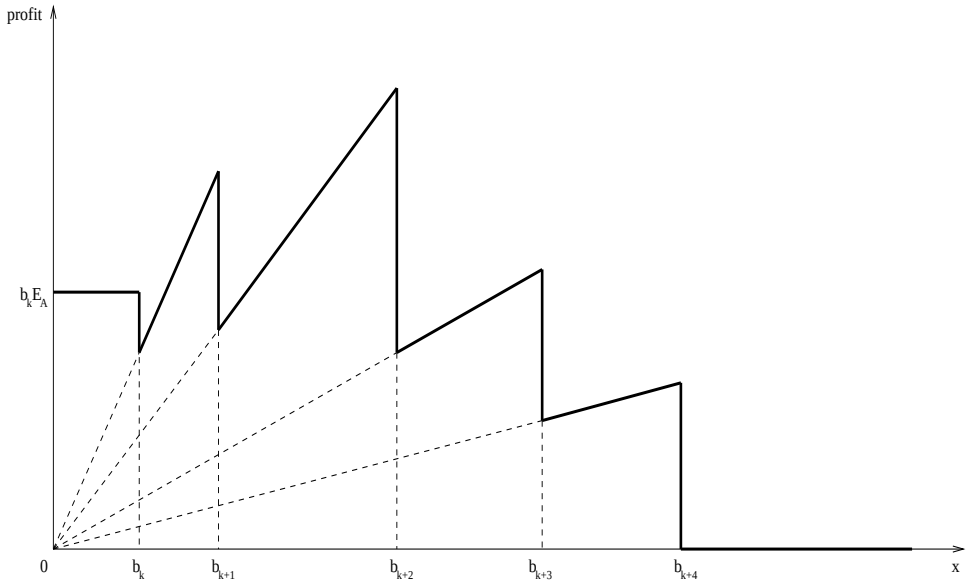


Figure 3.2: The dependence of the profit of Company A on its bid x.

the profit can "jump" across different contours. The profit z, projected onto the x-z plane, should actually be the piecewise linear profit curve in Figure 3.2.

Now we study the objective of the strong duality reformulation of the strategic bidding problem. As expected, this formulation is also a hard problem.

At the first glance, (3.6) is a problem with quadratic objective function, one quadratic constraint and some linear constraints, all of which are continuous and even differentiable. Especially, the objective function seems to be continuous.

However, we note that continuous objective function and constraints do not necessarily indicate continuous set of feasible points. Figure 3.3 shows such an example. Let a, b, c and d be linear inequality constraints, e be a nonlinear equality constraint and f be a linear equality constraint. We see that the only feasible points are P and P'. Hence the set of feasible points is discontinuous. It is possible that P and P' lie on different contours of the objective function, therefore the objective function could be discontinuous.

Numerically, the jumping value of the objective function can be observed if we solve

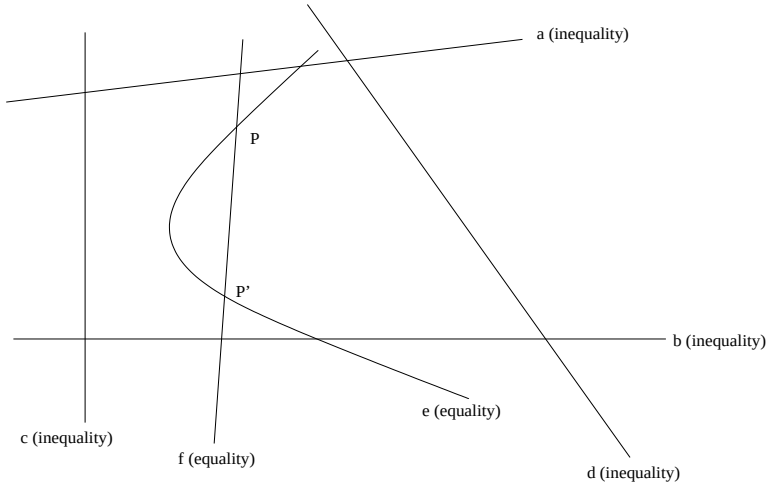


Figure 3.3: Discontinuous feasible set when all constraints are continuous.

(3.6) using a nonlinear programming solver, such as KNITRO. In the following run from KNITRO/DIRECT, the objective function value shows dramatic changes with a non-monotone manner in the cited iterations. This also happens in many other runs.

```
 Iter(maj/min)  Res   Objective      Feas err    Opt err     ||Step||    CG its
--------------- ---  -------------  ----------  ----------  ----------  -------
      0/      0 ---  -8.524395e+02   5.154e+04
      1/      1 Acc  -8.566778e+02   5.107e+04   4.777e+01   4.252e+01        0
.............
      5/      5 Acc  -7.478716e+02   4.573e+04   4.077e+01   5.907e+02        0
      6/      6 Acc  -4.350291e+03   4.500e+04   3.824e+01   9.346e+01        6
      7/      7 Acc  -8.684500e+02   4.487e+04   3.808e+01   1.619e+01        3
      8/      8 Acc  -2.076703e+02   4.474e+04   3.793e+01   1.263e+01        7
      9/      9 Acc   9.741726e+00   4.454e+04   3.195e+01   1.880e+01        4
     10/     10 Acc  -2.031360e+03   4.398e+04   3.161e+01   4.955e+01        5
     11/     11 Acc  -2.694611e+02   4.386e+04   3.146e+01   1.277e+01        8
     12/     12 Acc  -3.931478e+03   4.302e+04   3.023e+01   7.444e+01        6
     13/     13 Acc  -3.769676e+02   4.290e+04   3.021e+01   1.841e+01        2
     14/     14 Acc   7.141958e+02   4.279e+04   3.007e+01   1.143e+01       16
     15/     15 Acc   1.030830e+03   4.254e+04   2.890e+01   1.980e+02        0
.............
    106/    106 Acc   5.169720e+05   8.365e+03   8.385e+01   2.841e+01        2
    107/    107 Acc   4.981375e+05   8.225e+03   8.436e+01   3.649e+01        2
```

```
    108/    108 Acc   5.022549e+05   8.192e+03   8.403e+01   3.496e+01        4
.............
    314/    314 Acc   8.038441e+05   1.778e+02   1.148e+02   8.296e-01        0
    315/    315 Acc   7.688233e+05   1.493e+02   2.683e+02   1.053e+03        0
    316/    316 Acc   7.692100e+05   1.493e+02   2.668e+02   1.038e+02        0
.............
    418/    418 Acc   5.278114e+05   1.024e+01   1.024e+01   3.148e+00        0
    419/    419 Acc   4.998244e+05   6.737e+00   3.310e+01   2.716e+03        0
    420/    420 Acc   4.996839e+05   6.718e+00   6.718e+00   2.212e+01        0
.............
    468/    470 Acc   4.051810e+05   9.106e-08   2.663e-05   2.838e+01        7

EXIT: LOCALLY OPTIMAL SOLUTION FOUND.

Final Statistics
----------------
Final objective value               =   4.05180999137220e+05
Final feasibility error (abs / rel) =   9.11e-08 / 1.77e-12
Final optimality error  (abs / rel) =   2.66e-05 / 2.70e-08
# of iterations (major / minor)     =        468 /      470
# of function evaluations           =        483
# of gradient evaluations           =        469
# of Hessian evaluations            =        468
Total program time (secs)           =       6.77338 (     2.560 CPU time)
```

3.4.2 Objective of the General Bilevel Program

It is well known that the objective function of a general bilevel program could be discontinuous on its feasible region. The following example illustrates that a bilevel program of the form (3.8) could have a discontinuous objective function because its feasible set could be discontinuous, even if the bilevel program is a bilevel linear program, that is, the lower level problem and the upper level problem are both linear.

Example 3.5 *(A two-dimensional bilevel linear program)*

Consider the following linear program in terms of x:

$$\underset{x}{\text{minimize}} \quad -x \tag{3.28a}$$

$$x - 2y \leq 0 \tag{3.28b}$$

$$x + y \leq 6 \tag{3.28c}$$

$$x \geq 0. \tag{3.28d}$$

Note that y is the given data for this problem. For each fixed value of y, we look for the maximum possible value of x such that constraints (3.28b)-(3.28d) are satisfied. The solution set of this problem is marked by the bold line segments AB and BO in Figure 3.4(a).

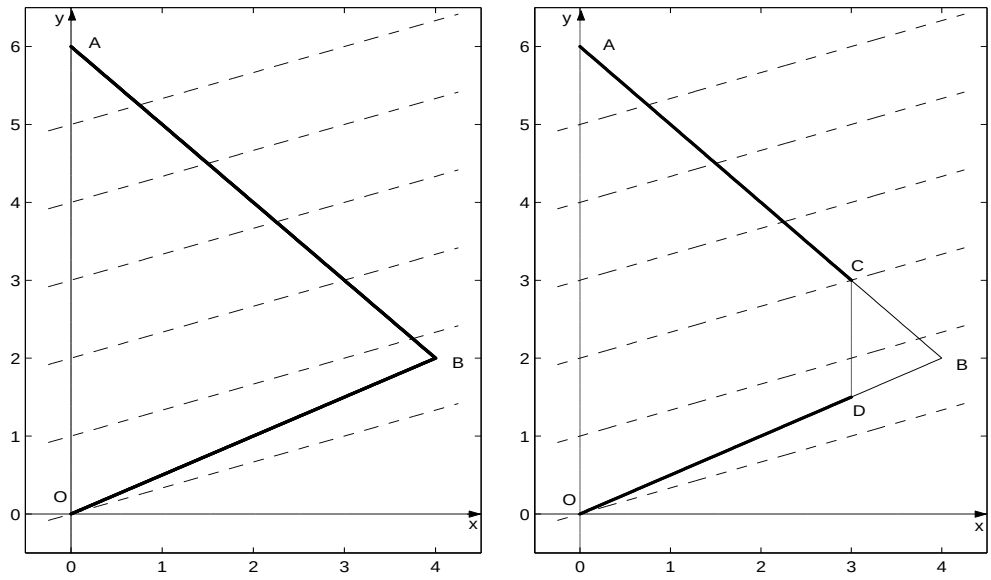
Let (3.28) be the lower level problem of the following bilevel program:

$$\underset{x,y}{\text{minimize}} \quad -x + 3y \tag{3.29a}$$

$$\text{subject to} \quad 0 \leq y \leq 6 \tag{3.29b}$$

$$x \text{ solves (3.28).} \tag{3.29c}$$

In Figure 3.4(a), it is seen that the outer level constraint (3.29b) does not interfere with the solution set of the lower level problem (segments AB and BO). So the feasible set of the bilevel problem (3.29) is still the bold line segments AB and BO, and it is continuous. The contours of the objective function $-x + 3y$ are shown in the figure by the dashed lines. We see then the optimal solution is $O(0, 0)$, which is both local and global.



(a) Graph for problem (3.29). (b) Graph for problem (3.30).

Figure 3.4: Feasible sets of two bilevel linear programs.

Let us consider another bilevel problem of the form

$$\underset{x,y}{\text{minimize}} \quad -x+3y \tag{3.30a}$$

$$\text{subject to} \quad 0 \le y \le 6 \tag{3.30b}$$

$$x \le 3 \tag{3.30c}$$

$$x \text{ solves (3.28).} \tag{3.30d}$$

The only difference between (3.29) and (3.30) is the constraint (3.30c). However, this constraint will interfere with the solution set of the lower level problem AB and BO. We can easily see from Figure (3.4)(b) that the feasible set of the bilevel program (3.30) is given by the bold segments AC and DO, which indicates that (3.30) has a discontinuous feasible set. Therefore this bilevel problem has two local solutions $(3,3)$ and $(0,0)$, of which the latter is the global solution. □

We conclude from this example that the discontinuous feasible set for a bilevel program is caused by the fact that it has a lower level optimization problem as part of the constraints, and some outer level constraints may restrain part of the solution set of the lower level problem from being feasible for the bilevel problem. The discontinuity is not dependent on the reformulation techniques we may use to generate a single level problem, such as the strong duality condition. Discontinuous feasible set often implies that bilevel programs have more than one local solutions, which make them difficult for general nonlinear programming solvers. In fact, it has been shown in Vicente, Savard and Judice [41] that finding the local optimal solution of a bilevel linear program is an NP-hard problem.

3.5 Allowing Uncertainty in Strategic Bidding

In earlier sections the properties of the strategic bidding model (3.4) are analyzed, where we assume that the electricity demand of the market and the bids of other companies are known. In practice this is not very likely to happen, and the strategic bidding model (3.4) is artificial. In this section we will slightly relax this assumption and improve the model by allowing uncertainty in the demand and other companies' price bids. The improved model is called the strategic bidding problem under uncertainty.

We consider a finite number of scenarios indexed by $s = 1, 2, \cdots, S$ and the probability of the occurrence of the s-th scenario is given by $p^s \in [0, 1]$. In each scenario we allow for a market demand d^s, a production capacity vector E^s, which we assume is known in the Bertrand model, and a vector of the price bids b^s from generators outside Company A. Accordingly, the production schedule, the spot price and the multipliers associated with the lower and upper bounds of the production schedule are also distinguished by scenarios. We denote these variables by y^s, λ^s, π_l^s and π_u^s, respectively. It is noted that only one vector of bids x from Company A is allowed, even for different scenarios. We set up the

strategic bidding model by maximizing Company A's average total profit over its bid, with the main constraints being that the independent operator determines the production schedule by solving the economic dispatch problem in each scenario. In other words, we would like to find the optimal bids for Company A such that the expected total profit for all different scenarios is maximized.

The economic dispatch problem for the s-th scenario ($1 \leq s \leq S$) is

$$\underset{y^s}{\text{minimize}} \quad \begin{bmatrix} x \\ b^s \end{bmatrix}^T y^s \tag{3.31a}$$

$$\text{subject to} \quad (y^s)^T e \geq d^s \quad (\text{multiplier: } \lambda^s) \tag{3.31b}$$

$$0 \leq y^s \leq E^s \quad (\text{multiplier: } \pi_l^s, \pi_u^s). \tag{3.31c}$$

Then the stochastic version of the bilevel strategic bidding problem is

$$\underset{x, y^s, \lambda^s, \pi_l^s, \pi_u^s}{\text{maximize}} \quad \sum_{s=1}^{S} p_s \left(\lambda^s e(1:t) - c_1\right)^T y^s(1:t) \tag{3.32a}$$

$$\text{subject to} \quad x \geq c_1 \tag{3.32b}$$

$$y^s \text{ solves (3.31) with multipliers } \lambda^s, \pi_l^s, \pi_u^s. \tag{3.32c}$$

We can again reformulate this bilevel optimization problem into a single level nonlinear

program. The stochastic version of the MPCC formulation of problem (3.32) is

$$\underset{x,y^s,\lambda^s,\pi_l^s,\pi_u^s}{\text{maximize}} \quad \sum_{s=1}^{S} p_s \left(\lambda^s e(1:t) - c_1\right)^T y^s(1:t) \tag{3.33a}$$

$$\text{subject to} \quad x \geq c_1 \tag{3.33b}$$

$$\lambda^s e - \pi_u^s + \pi_l^s = \begin{bmatrix} x \\ b^s \end{bmatrix} \quad (\forall 1 \leq s \leq S) \tag{3.33c}$$

$$0 \leq \lambda^s \perp \left((y^s)^T e - d^s\right) \geq 0 \quad (\forall 1 \leq s \leq S) \tag{3.33d}$$

$$0 \leq \pi_l^s \perp y^s \geq 0 \quad (\forall 1 \leq s \leq S) \tag{3.33e}$$

$$0 \leq \pi_u^s \perp (E^s - y^s) \geq 0 \quad (\forall 1 \leq s \leq S). \tag{3.33f}$$

The stochastic version of the strong duality formulation of problem (3.32) is

$$\underset{x,y^s,\lambda^s,\pi_l^s,\pi_u^s}{\text{maximize}} \quad \sum_{s=1}^{S} p_s \left(\lambda^s e(1:t) - c_1\right)^T y^s(1:t) \tag{3.34a}$$

$$\text{subject to} \quad x \geq c_1 \tag{3.34b}$$

$$(y^s)^T e \geq d^s \quad (\forall 1 \leq s \leq S) \tag{3.34c}$$

$$0 \leq y^s \leq E^s \quad (\forall 1 \leq s \leq S) \tag{3.34d}$$

$$\lambda^s e - \pi_u^s + \pi_l^s = \begin{bmatrix} x \\ b^s \end{bmatrix} \quad (\forall 1 \leq s \leq S) \tag{3.34e}$$

$$\lambda^s \geq 0, \pi_l^s \geq 0, \pi_u^s \geq 0 \quad (\forall 1 \leq s \leq S) \tag{3.34f}$$

$$\begin{bmatrix} x \\ b^s \end{bmatrix}^T y^s = d^s \lambda^s - (E^s)^T \pi_u^s \quad (\forall 1 \leq s \leq S). \tag{3.34g}$$

We note that the stochastic versions of the nonlinear programming reformulations of the

strategic bidding problem under uncertainty keep the basic structure of their deterministic counterparts (3.5) and (3.6), but the sizes of problems (3.33) and (3.34) can grow rapidly with the number of scenarios S. For a modest number of scenarios, they can be solved directly by modern nonlinear programming solvers. When the number of scenarios is large, the numerical solution of these problems would require special techniques such as decomposition techniques, which is beyond the range of this thesis.

3.6 Numerical Experiments with the Strong Duality Formulation

In this section we discuss some numerical experience with the strong duality formulation of the strategic bidding problem.

3.6.1 Numerical Experiments with One Scenario

We first solve the strong duality formulation using data from only one scenario using KNITRO and observe the following from the results:

- KNITRO/DIRECT: The problem is solved in 468/470 (major/minor) iterations, and the final objective value is 4.052×10^5. In 337 minor iterations the solver has to revert to KNITRO/CG because the KKT matrix has wrong inertia or the direct step is rejected. Among these CG steps, 70 of them report negative curvature, 250 of them report trust region bound crossed, and only 17 of them are exact steps. In terms of the steps, in the 133 minor iterations where direct steps are taken, the length of the step ($\|d\|$) ranges from 10^0 to 10^8 (steps are very large when close to the solution), but the slack bound cuts most of these long steps (α_s ranges from 10^{-5} to 10^{-1}); in the 337 minor iterations

where CG steps are taken, $\|d\|$ is usually 10^1 to 10^2, and many CG steps reach the trust region bound even if the trust region radius is large.

- KNITRO/CG: The problem is solved in 311/314 (major/minor) iterations, and the final objective value is 4.052×10^5. Among all the 314 minor iterations, 7 of them report negative curvature, 197 of them report trust region bound crossed, and 110 of them are exact steps. $\|d\|$ ranges from 10^0 to 10^3.

- KNITRO/ACTIVE: The problem is solved in 39/40 (major/minor) iterations, but the final objective value is 2.332×10^5. In the EQP phase, in most of the iterations the solver reports negative curvature after the first CG iteration.

We see that the interior point solvers KNITRO/DIRECT and KNITRO/CG perform poorly on this problem. Our observations could suggest some possibilities: the Hessian of Lagrangian is singular, LICQ fails, or multiple local solutions are present.

In order to counteract the effect of singular Hessians, we regularize the problem by adding a term of the form $-\rho X^T X$ to the objective function (3.6a) (since this is a maximization problem), where $\rho = 10^{-6}$, and X is the vector of all variables $(x, y, \lambda, \pi_l, \pi_u)$. However, this regularization technique only brings marginal improvements in the performance of the solvers.

In order to look for explicit evidence of the failure of constraint qualification conditions, we use KNITRO to solve problem (3.6) (with regularization) to a very high accuracy (with KNITRO option `opttol=5.0e-9`). It takes KNITRO/DIRECT 474/541 (major/minor) iterations to achieve the desired accuracy, but when close to the solution (beyond iteration 448), there are more than 30 minor iterations in which the solver only takes 1 CG iteration to the trust region bound. By evaluating the constraint values and their gradients at every iterate of the run, we observe that the Jacobian matrix of the active constraints is rank deficient

close to the optimal solution. From major iteration 442 the smallest singular value of the Jacobian of the active constraints is of order 10^{-15}, which suggests that there are linearly dependent constraint gradients. Therefore we know that LICQ fails at these iterates of the run.

3.6.2 Numerical Experiments with More Scenarios

We allow uncertainty in the strategic bidding model and solve the strong duality formulation (3.34) using data from 10 scenarios ($S = 10$) with equal possibilities ($p_s = 0.1$ for all $1 \leq s \leq 10$). We also use the `multistart` option in KNITRO and test more than one starting points with the hope to find a good local solution. Table 3.1 reports the performance of the solvers.

We observe from the numerical results that the problem is still difficult for all algorithms implemented in KNITRO. When only one starting point is used, KNITRO/DIRECT gives the best objective value; when more starting points are used, KNITRO/CG gives another local solution with better objective value. This shows that the multistart heuristic helps to find better solutions of the strategic bidding problem. The CPU time for finding a good solution is high but acceptable.

solver	#starts	final objective	CPU time (sec)
KNITRO/DIRECT	1	5.44957×10^5	245.360
	3	5.44957×10^5	1937.880
	10	5.44957×10^5	8281.650
KNITRO/CG	1	5.27291×10^5	495.360
	3	5.48548×10^5	1729.050
	10	5.48548×10^5	3153.160
KNITRO/ACTIVE	1	4.78499×10^5	2.950
	3	4.78499×10^5	7.690
	10	4.78499×10^5	24.380

Table 3.1: Performance of KNITRO on the strong duality formulation of the strategic bidding problem with 10 scenarios.

Chapter 4

Step Acceptance Comparison

Filter techniques have been proposed as a way to promote fast convergence of optimization algorithms. They are designed to replace exact penalty functions, which require the appropriate choice of a penalty parameter. The stated goal of filter techniques is to interfere as little as possible with Newton steps. This is a worthy goal as it has been observed that unrestricted Newton steps often result in fast convergence even far away from the solution. There are, however, some conceptual drawbacks in the way filters are defined (see Curtis [10]). In particular, as a filter is constructed based on previous function and constraint values, steps that result in an increase in the objective function may be prohibited, even if they give rise to a reduction in the constraints. Contemporary penalty techniques adjust the penalty parameter to allow these kinds of steps.

The goal of this study is to observe whether the potential drawbacks of filter techniques can be observed in practice. To this end, we made some controlled experiments with two codes, both of the trust region type. The first code is FILTER, which implements an active-set SQP method and uses a filter globalization technique (Fletcher and Leyffer [16, 15]). The second code is KNITRO/CG, an interior point method that uses an exact penalty function for step acceptance. Our goal is to measure the frequency with which each code accepts a full

step.

We selected 291 small scale constrained optimization problems from the CUTEr collection in Ampl format. These problems were selected so that the sum of the number of variables and the number of constraints is no larger than 500. The performance profile in Figure 4.1 shows that the two solvers have a similar level of robustness. As expected, FILTER tends to require fewer iterations because an active-set algorithm tends to be very effective on small problems.

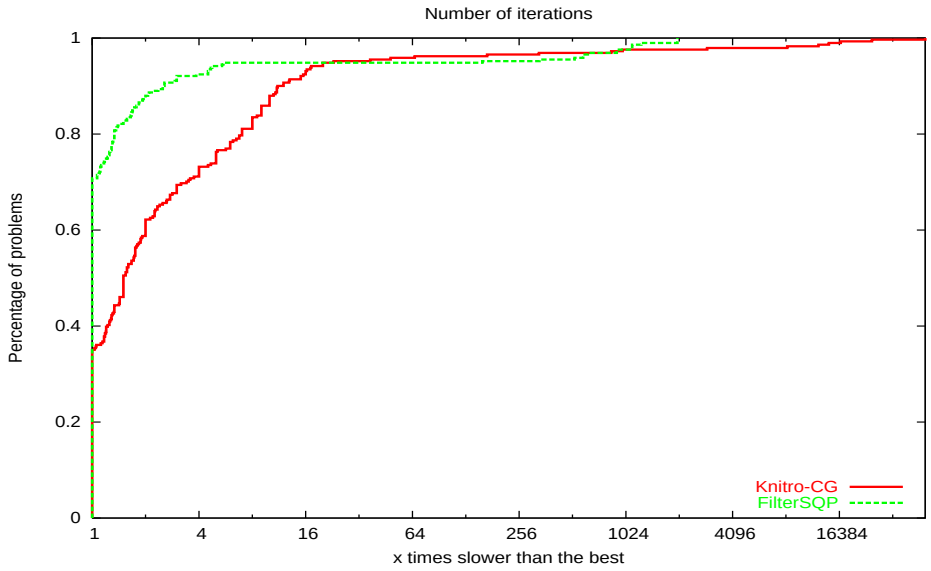


Figure 4.1: Performance profile for KNITRO-CG and FILTER on 291 small constrained problems, number of iterations.

Since both solvers implement a trust region approach, the computed step in each minor iteration is not subject to a line search, so the full step is judged by the penalty function or the filter. If the step is rejected, a second order correction (SOC) step is computed and the corrected step is examined by the penalty function or the filter again. There are 230 problems from the set of 291 problems for each of which both solvers terminate with optimality in 1000 iterations and report the same optimal objective function value. We compare the step acceptance rate of the two solvers for these 230 problems.

In our comparison we will consider two cases:

1. When second order correction steps are not counted: In this case we will only consider the Newton step computed in each iteration.

   For KNITRO/CG, the total number of iterations is 6883, and the total number of accepted steps for which SOC steps are not performed is 5080. So the step acceptance rate is $5080/6883 = 73.81\%$.

   For FILTER, the total number of iterations is 2784, the total number of feasibility restoration iterations is 295, and the total number of accepted steps for which SOC steps are not incurred is 1666. So the step acceptance rate is $1666/(2784 - 295) = 66.93\%$.

2. When SOC steps are counted: In this case we consider the Newton step with possible second order corrections in each iteration.

   For KNITRO/CG, the total number of iterations is 6883, and the total number of accepted steps is 5439. So the step acceptance rate is $5439/6883 = 79.02\%$. There are 446 iterations distributed in 50 problems, where negative curvature is reported, of which there are 216 accepted steps with negative curvature, so if we only consider the steps with possible SOC but without negative curvature, the step acceptance rate is $(5439 - 216)/6883 = 75.88\%$.

   For FILTER, the total number of iterations is 2784, the total number of feasibility restoration iterations is 295, the total number of accepted steps for which SOC steps are not incurred is 1666, and the total number of accepted steps for which SOC steps are incurred is 223. So the step acceptance rate is $(1666+223)/(2784-295) = 75.89\%$.

In summary, by none of these measures is the filter mechanism more tolerant than the penalty function. To our knowledge, this is the first time that such an observation has

been made. One should make some qualifications. First of all, our experiments were done using two different algorithms and codes. A more accurate approach would be to try to implement a filter and a penalty function in the same algorithm. This is not easy, however, as filter techniques usually include a feasibility restoration phase that significantly alters the underlying algorithm. A second observation is that the step acceptance mechanism should accept as many productive steps as possible, not just any steps. This is difficult to measure.

However, we are able to make more observations by looking closely at the data for the FILTER code. There are 52 problems for which FILTER rejects Newton steps. Let f denote the function value and $\|c\|$ denote the constraint violation. For every computed step, we divide the f-$\|c\|$ space into four regions:

1. Region 1: f increases and $\|c\|$ increases;
2. Region 2: f decreases and $\|c\|$ increases;
3. Region 3: f decreases and $\|c\|$ decreases;
4. Region 4: f increases and $\|c\|$ decreases.

Steps into Region 3 will be accepted by both step acceptance techniques. Therefore we consider the proportion of steps into the other regions. For each the 52 problems, we calculate the percentages of rejected Newton steps falling into each of the four regions described above. After that we calculate the averages of the percentages of rejected steps into these regions. We find that on average the percentages of rejected steps into Regions 1, 2 and 4 are 49.7%, 27.3% and 21.78%, respectively.

As suspected, the filter mechanism is rejecting a significant proportion of steps (21%) in Region 4. These can be valuable steps, as it can be useful to allow an increase in the objective function in order to approach the feasible region. A contemporary penalty method

often permits these steps and we speculate that an important weakness of filter methods is its inability to do so in some circumstances.

Bibliography

[1] G. Anandalingam and D. White, *A solution method for the linear static stackelberg problem using penalty functions*, IEEE Transactions on Automatic Control, 35 (1990), pp. 1770–1773.

[2] J. F. Bard, *Practical Bilevel Optimization: Algorithms and Applications*, Springer Series in Nonconvex Optimization and Its Applications, Vol. 30, Springer Verlag, Heidelberg, Berlin, New York, 1999.

[3] L. Bergamaschi, J. Gondzio, and G. Zilli, *Preconditioning indefinite systems in interior point methods for optimization*, Tech. Rep. MS-02-002, Department of Mathematics and Statistics, University of Edinburgh, Scotland, 2002.

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

[5] R. H. Byrd, *Robust trust region methods for constrained optimization.* Third SIAM Conference on Optimization, Houston, Texas, May 1987.

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

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

[8] R. H. Byrd, J. Nocedal, and R. Waltz, *KNITRO: An integrated package for nonlinear optimization*, in Large-Scale Nonlinear Optimization, G. di Pillo and M. Roma, eds., Springer, 2006, pp. 35–59.

[9] T. F. Coleman and A. Verma, *A preconditioned conjugate gradient approach to linear equality constrained minimization*, tech. rep., Computer Science Department and Cornell Theory Center, Cornell University, Ithaca, NY 14850, USA, July 1998.

[10] F. E. Curtis, *Inexact Sequential Quadratic Programming Methods for Large-Scale Nonlinear Optimization*, PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois, USA, 2007.

[11] S. Dempe, *Foundations of Bilevel Programming*, Springer Series in Nonconvex Optimization and Its Applications, Vol. 61, Springer Verlag, Heidelberg, Berlin, New York, 2002.

[12] H. S. Dollar, N. I. M. Gould, and A. J. Wathen, *On implicit-factorization constraint preconditioners*, Tech. Rep. RAL-TR-2004-036, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2004.

[13] I. S. Duff, *MA57 – a code for the solution of sparse symmetric definite and indefinite systems*, ACM Transactions on Mathematical Software, 30 (2004), pp. 118–144.

[14] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Transactions on Mathematical Software, 9 (1983), pp. 302–325.

[15] R. Fletcher and S. Leyffer, *User manual for filterSQP*, Numerical Analysis Report NA/181, Department of Mathematics, University of Dundee, Dundee, Scotland, 1998.

[16] ——, *Nonlinear programming without a penalty function*, Mathematical Programming, 91 (2002), pp. 239–269.

[17] A. Forsgren, P. E. Gill, and J. D. Griffin, *Iterative solution of augmented systems arising in interior methods*, Tech. Rep. NA 05-3, Department of Mathematics, University of California, San Diego, 2005.

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

[19] D. Fudenberg and J. Tirole, *Game Theory*, MIT Press, 1996. Fifth printing.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, second ed., 1989.

[21] N. I. M. Gould, M. E. Hribar, and J. Nocedal, *On the solution of equality constrained quadratic problems arising in optimization*, SIAM Journal on Scientific Computing, 23 (2001), pp. 1375–1394.

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

[23] N. I. M. Gould, D. Orban, and P. L. Toint, *Numerical methods for large-scale nonlinear optimization*, Technical Report RAL-TR-2004-032, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2004.

[24] L. Hei, J. Nocedal, and R. A. Waltz, *A numerical study of active-set and interior-point methods for bound constrained optimization*, Technical Report, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois, USA, June 2006. Submitted for publication.

[25] ——, *On constraint preconditioners for interior point algorithms in general nonlinear programming*. Working paper, 2007.

[26] B. F. Hobbs, C. B. Metzler, and J.-S. Pang, *Strategic gaming analysis for electric power systems: An MPEC approach*, IEEE Transactions on Power Systems, 15 (2000), pp. 638–645.

[27] C. A. Johnson and A. Sofer, *A primal-dual method for large-scale image reconstruction in emission tomography*, SIAM Journal on Optimization, 11 (2001), pp. 691–715.

[28] C. Keller, N. I. M. Gould, and A. J. Wathen, *Constraint preconditioning for indefinite linear systems*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1300–1317.

[29] S. Leyffer, G. López-Calva, and J. Nocedal, *Interior methods for mathematical programs with complementarity constraints*, SIAM Journal on Optimization, 17 (2006), pp. 52–77.

[30] C. J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM Journal on Scientific Computing, 21 (1999), pp. 24–45.

[31] Z. Lu, R. D. C. Monteiro, and J. W. O'Neal, *An iterative solver-based infeasible primal-dual path-following algorithm for convex quadratic programming*, SIAM Journal on Optimization, 17 (2006), pp. 287–310.

[32] J. L. Morales and J. Nocedal, *Automatic preconditioning by limited memory quasi-newton updating*, ACM Transactions on Mathematical Software, 10 (2000), pp. 1079–1096.

[33] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer, second ed., 2006.

[34] E. O. Omojokun, *Trust region algorithms for optimization with nonlinear equality and inequality constraints*, PhD thesis, University of Colorado, Boulder, Colorado, USA, 1989.

[35] M. V. Pereira, S. Granville, R. Dix, and L. A. Barroso, *Strategic bidding under uncertainty: A comparison between the binary expansion approach and nonlinear optimization methods*, tech. rep., PSR Inc., 2004.

[36] M. V. Pereira, S. Granville, M. H. C. Fampa, F. Dix, and L. A. Barroso, *Strategic bidding under uncertainty: A binary expansion approach*, IEEE Transactions on Power Systems, 20 (2005), pp. 180–188.

[37] M. Roma, *Dynamic scaling based preconditioning for truncated Newton methods in large scale unconstrained optimization: The complete results*, Technical Report R. 579, Istituto di Analisi dei Sistemi ed Informatica, 2003.

[38] H. Scheel and S. Scholtes, *Mathematical programs with complementarity constraints: Stationarity, optimality and sensitivity*, Mathematics of Operations Research, 25 (2000), pp. 1–22.

[39] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Journal of Future Generation Computer Systems, 20 (2004), pp. 475–487.

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

[41] L. Vicente, G. Savard, and J. Judice, *Descent approaches for quadratic bilevel programming*, Journal of Optimization Theory and Applications, 81 (1994), pp. 379–399.

[42] A. Wächter, *An interior point algorithm for large-scale nonlinear optimization with applications in process engineering*, PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 2002.

[43] A. Wächter and L. T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Mathematical Programming, 106 (2006), pp. 25–57.

[44] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban, *An interior algorithm for nonlinear optimization that combines line search and trust region steps*, Mathematical Programming, Series A, 107 (2006), pp. 391–408.

[45] J. D. Weber and T. J. Overbye, *An individual welfare maximization algorithm for electricity markets*, IEEE Transactions on Power Systems, 17 (2002), pp. 590–596.

[46] D. White and G. Anandalingam, *A penalty function approach for solving bi-level linear programs*, Journal of Global Optimization, 3 (1993), pp. 397–419.