2014

A Subproblem Algorithm for the Adaptive Augmented Lagrangian Method

Wenda Zhang
Lehigh University

A Subproblem Algorithm for the Adaptive Augmented Lagrangian Method

by

Wenda Zhang

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial and Systems Engineering

Lehigh University

April 2014

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Master of Science.

_____
Date

_____
Frank E. Curtis, Thesis Advisor

_____
Chairperson of Department

Acknowledgements

I'd like to express my deepest appreciation to Professor Frank E. Curtis for the instructive guidance throughout the work on the thesis. He provided me new ideas to explore and, more importantly, a systematic and rational way of exploring them. Without his advice, this thesis would not have been possible. I thank Zheng Han for his aid on the numerical experiments. I'd also like to give special thanks to Mr. Douglas Adams who gave, in my opinion, the best advice to anybody facing an unknown or difficult situation, whether it's hitchhiking in the galaxy, or trying to write a thesis: Don't panic.

Contents

List of Tables

List of Figures

Abstract

An adaptive augmented Lagrangian algorithm is presented to overcome some undesirable behavior of traditional augmented Lagrangian methods. While the method has previously been proposed, the goal in this thesis is to improve its practical performance. In particular, we propose an active set projected conjugate gradient (ASPCG) method for solving the subproblems of the adaptive augmented Lagrangian algorithm. The proposed ASPCG algorithm first estimates the optimal active set and then performs a projected conjugate gradient method to produce the exact or at least a good approximate solution updating the active set estimate when appropriate. We perform a series of numerical experiments to determine if the proposed algorithm is superior in some critical performance measures to the solver originally implemented in the adaptive augmented Lagrangian algorithm. In addition, we conduct experiments to monitor the performance of the adaptive augmented Lagrangian algorithm when some of its key features are modified.

Chapter 1

Introduction

The method of multipliers, or the augmented Lagrangian (AL) method, is an important algorithm for solving constrained optimization problems. However, basic AL approaches share one critical disadvantage when they are used to solve nonlinear problems: they can perform poorly if the initial choices of the penalty parameter or Lagrange multipliers is poor. If the penalty parameter is too large or the estimates of the optimal multipliers are not close to the optimal ones, the algorithm may be stalled at a point with little or no progress made towards a solution. An adaptive AL (AAL) method has been proposed by Curtis, Jiang and Robinson [2] to address this disadvantage. In the adaptive AL method, it is required that each trial step yields a sufficiently large reduction in linearized constraint violation, or the penalty parameter will be decreased to place a higher emphasis on minimizing the constraint violation. This procedure ensures the progress towards constraint satisfaction, even when the current iterate is far away from the feasible region.

An important part of the AAL algorithm is the computation of the steering step and trial step from their respective QP subproblems. Here, the subproblems can be treated as two very similar bound-constrained QPs, and they can be solved by the same algorithm. The quality of the solutions and the efficiency of computing the solutions affect directly the performance of the AAL algorithm. The purpose of this thesis is to present a viable method for solving the subproblems in the hope of improving the practical performance of the original AAL algorithm, and analyze its performance within the framework of the AAL algorithm under various conditions. We introduce an iterative strategy for estimating the optimal active set [6] for the bound-constrained QP with

a projected conjugate gradient (projected CG) method [3, 7]. The projected CG method allows elements to be added to the active set estimate quickly, and we also allow elements to exit the estimate based on violations of the KKT conditions. This should, in theory, produce an accurate estimate of the optimal active set so that the projected CG method could compute an accurate approximate solution for the subproblem.

The thesis is organized as follows. In Chapter 2, we outline a traditional and the adaptive AL algorithm, discuss the inefficiencies troubling the traditional method, and show how the new procedures in the adaptive method help to overcome such disadvantage. In Chapter 3, we present a projected CG method with procedures to estimate the optimal active set for the subproblem. In Chapter 4, we provide numerical results that compare the performance of AAL as implemented in [2] and traditional AL method with our proposed algorithm to show the effectiveness of our approach. We also explore the performance of AAL with our proposed algorithm when the meaning of a "sufficiently large" reduction in constraint violation is varied. Finally, in Chapter 5, we further analyze and discuss the proposed subproblem solver and the AAL algorithm as a whole.

Notation. Once a function is defined, we often drop the dependency on its arguments when using it again later. We use subscripts in these cases to denote the iteration number of an algorithm during which the function is used. For example, we use f_k instead of $f(x_k)$, where f is a defined function and x_k is the current primal iterate.

Chapter 2

An Adaptive Augmented Lagrangian Algorithm

A general nonlinear optimization problem can be converted to one with equality constraints and bound constraints

$$\min_{x\in\mathbb{R}^n} f(x) \ \text{ subject to } \ c(x)=0, \ \ l\le x\le u, \tag{2.1}$$

where we assume the objective function $f:\mathbb{R}^n\to\mathbb{R}$ and constraint function $c:\mathbb{R}^n\to\mathbb{R}^m$ to be twice continuously differentiable. A first-order optimal primal-dual solution of problem (2.1) is found by locating a pair (x,y) satisfying

$$\begin{cases} F_L(x,y)=x-P[x-\nabla_x\ell(x,y)]=x-P[x-(\nabla f(x)-\nabla c(x)y)]=0 \\ \nabla_y\ell(x,y)=-c(x)=0 \end{cases} \tag{2.2}$$

where $\ell(x,y)$ is the Lagrangian of problem (2.1) defined as

$$\ell(x,y):=f(x)-c(x)^Ty, \tag{2.3}$$

and P is a projection operator defined by

$$(P[x])_i = \begin{cases} l_i & \text{if } x_i \leq l_i, \\ u_i & \text{if } x_i \geq u_i, \\ x_i & \text{otherwise.} \end{cases} \tag{2.4}$$

However, it is not guaranteed that we can find a feasible solution, so if (2.1) is infeasible, it is commonly preferred for the algorithm to settle by finding a stationary point for $\frac{1}{2}\|c(x)\|_2^2$ subject to the bound constraints, which correspond to the problem

$$\min_{x \in \mathbb{R}^n} v(x) \;\text{ subject to }\; l \leq x \leq u, \;\text{ where }\; v(x) := \frac{1}{2}\|c(x)\|_2^2. \tag{2.5}$$

A first-order necessary condition for (2.5) is that a point x satisfies

$$F_{Feas}(x) := x - P[x - \nabla_x v(x)] = x - P[x - \nabla c(x) c(x)] = 0. \tag{2.6}$$

The point x is an infeasible stationary point for problem (2.1) if $v(x) > 0$ when (2.6) holds.

In this chapter, we discuss Augmented Lagrangian algorithms that focus on solving (2.1), or at least (2.5). Section 2.1 examines the procedures of a basic AL approach, and Section 2.2 describes an adaptive AL method that will be the basic framework for all the discussion later.

2.1 Basic Augmented Lagrangian Algorithm

The Augmented Lagrangian method combines the standard Lagrangian function ℓ with the quadratic penalty function. In terms of the problem we are trying to solve, we apply a bound-constrained Augmented Lagrangian approach, which incorporates only the equality constraints from (2.1) into the augmented Lagrangian, that is,

$$\mathcal{L}(x, y, \mu) := \mu \ell(x, y) + v(x) = \mu(f(x) - c(x)^T y) + \frac{1}{2}\|c(x)\|_2^2,$$

where μ is the penalty parameter scaling the Lagrangian function. In each iteration of the AL method, the penalty parameter μ is fixed at some value μ_k, the Lagrange multiplier vector y is fixed at an estimate y_k, and we minimize the augmented Lagrangian function $\mathcal{L}(x, y, \mu)$ with respect to x, the minimizer of which we denote as

$$x(y, \mu) := \underset{x \in \mathbb{R}^n}{\arg\min}\, \mathcal{L}(x, y, \mu) \;\text{ subject to }\; l \leq x \leq u. \tag{2.7}$$

If a solution for the problem in (2.7) can be found, it must satisfy the first-order optimality condition

$$F_{AL}(x, y, \mu) = P[x - \nabla_x \mathcal{L}(x, y, \mu)] - x = 0, \tag{2.8}$$

where $\nabla_x \mathcal{L}(x, y, \mu)$ is the gradient of the augmented Lagrangian with respect to x at (x, y, μ), that is,

$$\nabla_x \mathcal{L}(x, y, \mu) = \mu(\nabla f(x) - \nabla c(x)\pi(x, y, \mu)), \;\text{ where }\; \pi(x, y, \mu) := y - \frac{1}{\mu} c(x). \tag{2.9}$$

For a solution $x(y, \mu)$ of (2.7), notice that if $c(x(y, \mu)) = 0$ holds, then together with (2.8), we have (2.2) satisfied for the original problem. This suggests that such a solution $x(y, \mu)$ is a first-order optimal solution for (2.1). We now outline the structure of a basic AL algorithm in Algorithm 1.

The constraint violation $c(x)$ is checked when (2.7) is solved; if it is sufficiently small, the penalty parameter is not changed for the next iteration, the Lagrangian multiplier estimates are updated using the function π and the tolerance t is tightened; otherwise, we decrease the penalty parameter to place more emphasis on reducing constraint violation in subsequent iterations.

However, notice that while the penalty parameter should be updated when constraint violation is large, we do not have a good idea on what x_{k+1} should be in such a case. We can certainly update the primal point like we do if the constraint violation is small, but how much progress that can be achieved by setting $x_{k+1} \leftarrow x(y_k, \mu_k)$ is uncertain when the penalty parameter μ_k has been deemed large. We can also choose to maintain the current value $x_{k+1} = x_k$ and dismiss the potential progress that can be made by updating the primal point. In any case, the only sure progress made is the update of the penalty parameter.

Due to this flaw, the basic AL algorithm may not be very efficient at times, especially when

Algorithm 1 Basic Augmented Lagrangian Algorithm

Choose constants $\{\gamma_\mu, \gamma_t\} \subset (0,1)$, initial tolerance $t_0 > 0$ and initial penalty parameter $\mu_0 > 0$.
Choose initial point x_0 and multipliers y_0.
Set $k \leftarrow 0$, and $j \leftarrow 0$.
loop
    if (2.2) is satisfied, **then**
        return : the first-order optimal solution (x_k, y_k).
    if (2.6) and $v(x_k) > 0$ holds, **then**
        return : the infeasible stationary point x_k.
    Find a point $x(y_k, \mu_k)$ that satisfies (2.8).
    if $\|c(x(y_k, \mu_k))\| \leq t_j$, **then**
        Set $x_{k+1} \leftarrow x(y_k, \mu_k)$.
        Set $y_{k+1} \leftarrow \pi(x_{k+1}, y_k, \mu_k)$.
        Set $\mu_{k+1} \leftarrow \mu_k$.
        Set $t_{j+1} \leftarrow \gamma_t t_j$.
        Set $j \leftarrow j + 1$.
    else
        Choose $x_{k+1} \leftarrow x_k$ or $x_{k+1} \leftarrow x(y_k, \mu_k)$.
        Set $y_{k+1} \leftarrow \pi(x_{k+1}, y_k, \mu_k)$.
        Set $\mu_{k+1} \leftarrow \gamma_\mu \mu_k$.
    Set $k \leftarrow k + 1$.

the penalty parameter is too large or the Lagrange multiplier y is not an accurate estimation of the optimal multiplier. That's why we choose to apply an adaptive approach towards updating the penalty parameter and minimizing the objective function, which we will discuss in the next section.

2.2 An Adaptive AL Trust Region Approach

In this section, we present an Adaptive AL (AAL) trust region method that will be the main framework for the algorithms we discuss the Chapter 3, which are used to solve the subproblems in the AAL trust region approach. The main idea of this approach is to evaluate the reduction in constraint violation with a certain trial step against the reduction obtained by a step that solely tries to minimize constraint violation, a step that we call a feasibility step. If the reduction in the former case is not sufficiently large comparing with the latter case, and the current constraint violation is not so small that it doesn't matter if the trial step produces enough reduction, then the penalty parameter is decreased to place more emphasis on reducing constraint violation for

the next trial step.

To be specific, each iteration of the algorithm produces a trial step s_k by finding a step minimizing the augmented Lagrangian from the current primal point x_k. It is desirable for this step to make progress in terms of (2.1), and that means on top of minimizing the augmented Lagrangian, this step should be one reducing constraint violation sufficiently. As we explained earlier, a feasibility step r_k is thus computed before the trial step with the objective of minimizing constraint violation, to help us find a trial step s_k that makes enough progress on both the augmented Lagrangian and constraint violation.

Feasibility Step Subproblem

In order to compute the feasibility step, we first introduce an approximation function for the constraint violation measure, that is,

$$q_v(s, x) := \frac{1}{2}\|c(x) + \nabla c(x)s\|_2^2 \approx v(x + s).$$

The feasibility step is defined as the solution of the following quadratic trust-region subproblem

$$\min_{r\in\mathbb{R}^n} q_v(r, x_k) \;\text{ subject to }\; l \le x_k + r \le u, \;\; \|r\|_2 \le \theta_k, \tag{2.10}$$

where $\delta > 0$ is a constant and both

$$\theta_k := \theta(x_k, \delta_k) := \min\{\delta_k, \delta\|F_{Feas}(x_k)\|_2\} \ge 0 \tag{2.11}$$

and $\delta_k > 0$, the current trust region radius, are set dynamically within the algorithm. Equation (2.11) makes sure the trust region radius θ_k approaches zero as the algorithm approaches stationary points of the constraint violation measure, keeping the step from being too large when the constraint violation is small.

To ensure progress is made by a solution of the subproblem, we compute a Cauchy step r_k^C for the subproblem so that we can evaluate the progress made by a solution of (2.10). The Cauchy

step is defined as

$$r_k^C := P[x_k - \beta_k \nabla c_k c_k] - x_k, \tag{2.12}$$

where β_k is chosen so that the resulting Cauchy point produces a sufficient reduction in q_v. In this case, it is required that

$$\Delta q_v(r_k^C, x_k) := q_v(0, x_k) - q_v(r_k^C, x_k) \geq -\epsilon_r (r_k^C)^T \nabla c_k c_k \ \text{ and } \ \|r_k^C\|_2 \leq \theta_k \tag{2.13}$$

holds for some $\epsilon_r \in (0, 1)$. We now show the process of computing such a Cauchy step, as well as auxiliary nonnegative scalar quantities Γ_k and ϵ_k, which later will be used during the computation of the trial step s_k, in Algorithm 2.

Algorithm 2 Cauchy step computation for subproblem (2.10)

Require: $\theta_k \geq 0$.
1: Choose constants $\{\epsilon_r, \gamma\} \subset (0, 1)$.
2: Compute l_k as the smallest nonnegative integer that satisfies $\|P[x_k - \gamma^{l_k} \nabla c_k c_k] - x_k\|_2 \leq \theta_k$.
3: **if** $l_k > 0$, **then**
4: Set $\Gamma_k \leftarrow \min\{2, \frac{1}{2}(1 + \|P[x_k - \gamma^{l_k} \nabla c_k c_k] - x_k\|_2 / \theta_k)\}$.
5: **else**
6: Set $\Gamma_k \leftarrow 2$.
7: Set $\beta_k \leftarrow \gamma^{l_k}$, $\epsilon_k \leftarrow 0$, and compute r_k^C with (2.12).
8: **while** (2.13) is not satisfied, **do**
9: Set $\beta_k \leftarrow \gamma \beta_k$, and $\epsilon_k \leftarrow \max\{\epsilon_k, -\frac{\Delta q_v(r_k^C, x_k)}{(r_k^C)^T \nabla c_k c_k}\}$.
10: Compute r_k^C with (2.12).
11: **return** : $(r_k^C, \epsilon_k, \Gamma_k)$

Trial Step Subproblem

Now we can proceed to identify the subproblem that computes the trial step s_k. Again, we need first introduce the approximate function of the augmented Lagrangian, that is,

$$q_{\mathcal{L}}(s, x, y, \mu) := \mu q_\ell(s, x, y) + q_v(s, x) \approx \mathcal{L}(x + s, y, \mu),$$

where $q_\ell(s, x, y)$ is an approximation of the Lagrangian,

$$q_\ell(s, x, y) := \frac{1}{2} s^T (\nabla^2_{xx} \ell(x, y)) s + \nabla_x \ell(x, y)^T s + \ell(x, y) \approx \ell(x + s, y).$$

The trial step s_k is defined as the solution of the following quadratic trust region subproblem

$$\min_{s\in\mathbb{R}^n} q_{\mathcal{L}}(s, x_k, y_k, \mu_k) \text{ subject to } l \leq x_k + s \leq u, \ \ \|s\|_2 \leq \Theta_k, \tag{2.14}$$

where

$$\Theta_k := \Theta(x_k, y_k, \mu_k, \delta_k, \Gamma_k) := \Gamma_k \min\{\delta_k, \delta\|F_{AL}(x_k, y_k, \mu_k)\|_2\} \geq 0. \tag{2.15}$$

Similar to (2.10), the trust-region radius is set up this way so that when the algorithm approaches stationary points of the augmented Lagrangian, the radius goes to zero.

Also similar to the procedure of solving the feasibility step subproblem, we compute a Cauchy point, this time for (2.14), as

$$s_k^C := P[x_k - \alpha_k \nabla_x \mathcal{L}(x_k, y_k, \mu_k)] - x_k, \tag{2.16}$$

where $\alpha_k > 0$ is chosen so that the resulting Cauchy point produces a sufficient reduction in $q_{\mathcal{L}}$. In this case, it is required that

$$\Delta q_{\mathcal{L}}(s_k^C, x_k, y_k, \mu_k) \geq -\frac{\epsilon_r + \epsilon_k}{2}(s_k^C)^T \nabla_x \mathcal{L}(x_k, y_k, \mu_k) \text{ and } \|s_k^C\|_2 \leq \Theta_k \tag{2.17}$$

where

$$\Delta q_{\mathcal{L}}(s_k^C, x_k, y_k, \mu_k) := q(0, x_k, y_k, \mu_k) - q(s_k^C, x_k, y_k, \mu_k).$$

We can now show the procedure for computing the Cauchy step in Algorithm 3.

Algorithm 3 Cauchy step computation for subproblem (2.14)

1: Set constants $\{\epsilon_r, \gamma\} \subset (0, 1)$ and ϵ_k (Obtained from Algorithm 2).
2: Set $\alpha_k \leftarrow 1$ and compute s_k^C with (2.16).
3: **while** (2.17) is not satisfied, **do**
4: Set $\alpha_k \leftarrow \gamma\alpha_k$ and compute s_k^C with (2.16).
5: **return** : s_k^C

Notice that if we used the infinity norm to define the trust-region constraints of both subproblems, the trust-region constraints become bound constraints. In this case, both subproblems can be viewed as quadratic bound-constrained optimization problems. The two subproblems con-

stitute a very large part of the computational cost for the AAL algorithm; that is why we focus on the implementation of algorithms that can efficiently solve bound-constrained QPs, which we will be discussing in Chapter 3. For now, we are content with having the solutions of the two subproblems, a feasibility step r_k and a trial step s_k.

An Iteration of the Adaptive AL Trust Region Algorithm

We can now describe all the essential procedures of the AAL trust region algorithm, and show the entirety of it in Algorithm 4. Parts of an iteration of the AAL trust-region algorithm, the termination conditions, are very similar to that of the basic AL algorithm. For this reason, we will focus on the procedures that are not present in Algorithm 1.

The purpose of the **while** loop in line 10 is to detect if a feasible descent direction for $\mathcal{L}(\cdot, y_k, \mu_k)$ from x_k exists. When $F_{AL}(x_k, y_k, \mu_k) = 0$, then no such direction exists and we won't be able to compute a step towards a bound-constrained minimizer of the augmented Lagrangian $\mathcal{L}(\cdot, y_k, \mu_k)$.

Next, recall at the beginning of Section 2.2, we discussed the ideal trial step s_k for the AL algorithm, and the fundamental idea of how to find it. Now, with the Cauchy point for both subproblems found, we can present a set of conditions that the computed trial step must satisfy:

$$\Delta q_{\mathcal{L}}(s_k, x_k, y_k, \mu_k) \geq \kappa_1 \Delta q_{\mathcal{L}}(s_k^C, x_k, y_k, \mu_k) > 0, \tag{2.18a}$$

$$\Delta q_v(r_k, x_k) \geq \kappa_2 \Delta q_v(r_k^C, x_k), \tag{2.18b}$$

$$\Delta q_v(s_k, x_k) \geq \min\{\kappa_3 \Delta q_v(r_k, x_k), v_k - \frac{1}{2}(\kappa_t t_j)^2\}, \tag{2.18c}$$

where we define constants $\{\kappa_1, \kappa_2, \kappa_3, \kappa_t\} \subset (0, 1)$, and $t_j > 0$ is the tolerance of the constraint violation similar to that of Algorithm 1.

Equations (2.18a) and (2.18b) ensure the solutions we have yield at least some fraction of the progress that the two Cauchy points make, in constraint violation and the augmented Lagrangian, respectively. Equation (2.18c) ensures the trial step s_k yields a sufficient reduction in constraint violation compared to the feasibility step r_k. Notice that the second term of the right-hand side of (2.18c) can be very small or even negative, which in turn allows the right-hand side to be small or negative. This means that the trial step can produce a small decrease in constraint violation

Algorithm 4 Adaptive Augmented Lagrangian Trust Region Algorithm

Choose $\{\gamma, \gamma_\mu, \gamma_t, \gamma_T, \gamma_\delta, \kappa_1, \kappa_2, \kappa_3, \kappa_t, \epsilon_r, \eta_s\} \subset (0,1)$ and $\eta_{vs} \in (\eta_s, 1)$.
Choose $\{\delta, \delta_{min}, \delta_{max}, \epsilon\} \subset (0, \infty)$ and $\Gamma_\delta > 1$.
Choose (x_0, y_0) and $\{\mu_0, \delta_0, t_0, t_1, T_1\} \subset (0, \infty)$.
Set $k \leftarrow 0$, and $j \leftarrow 1$.
loop
  if (2.2) is satisfied, **then**
    return : the first-order optimal solution (x_k, y_k).
  if (2.6) and $v(x_k) > 0$ holds, **then**
    return : the infeasible stationary point x_k.
  while $F_{AL}(x_k, y_k, \mu_k) = 0$, **do**
    Set $\mu_k \leftarrow \gamma_\mu \mu_k$.
  Define θ_k by (2.11), and compute $(r_k^C, \epsilon_k, \Gamma_k)$.
  Define Θ_k by (2.15), and compute s_k^C.
  Compute solutions r_k to (2.10) that satisfies (2.18b), and s_k to (2.14) that satisfies (2.18a)
  while (2.18c) is not satisfied or $F_{AL}(x_k, y_k, \mu_k) = 0$, **do**
    Set $\mu_k \leftarrow \gamma_\mu \mu_k$ and define Θ_k by (2.15).
    Compute s_k^C, and s_k that satisfies (2.18a).
  Compute ρ_k from (2.19).
  if $\rho_k \geq \eta_{vs}$, **then**
    Set $x_{k+1} \leftarrow x_k + s_k$ and $\delta_{k+1} \leftarrow \min\{\delta_{max}, \Gamma_\delta \delta_k\}$.
  else if $\rho_k \geq \eta_s$, **then**
    Set $x_{k+1} \leftarrow x_k + s_k$ and $\delta_{k+1} \leftarrow \delta_k$.
  else
    Set $x_{k+1} \leftarrow x_k$ and $\delta_{k+1} \leftarrow \max\{\delta_{min}, \gamma_\delta \delta_k\}$.
  if $\|c_{k+1}\|_2 \leq t_j$, **then**
    if (2.20) is satisfied, **then**
      Set $\hat{y}_{k+1} \leftarrow \pi(x_{k+1}, y_k, \mu_k)$.
    else
      Set $\hat{y}_{k+1} \leftarrow y_k$.
    if $\min\{\|F_L(x_{k+1}, \hat{y}_{k+1})\|, \|F_{AL}(x_{k+1}, y_k, \mu_k)\} \leq T_j$, **then**
      Set $t_{j+1} \leftarrow \min\{\gamma_t t_j, t_j^{1+\epsilon}\}$ and $T_{j+1} \leftarrow \gamma_T T_j$.
      Set $y_{k+1} \leftarrow \hat{y}_{k+1}$.
      Set $j \leftarrow j + 1$.
    else
      Set $y_{k+1} \leftarrow y_k$.
  else
    Set $y_{k+1} \leftarrow y_k$.
  Set $\mu_{k+1} \leftarrow \mu_k$.
  Set $k \leftarrow k + 1$.

or even an increase, as long as the current constraint violation is sufficiently small relative to the tolerance t_j. If (2.18c) is not satisfied by the current s_k, we reduce the penalty parameter and compute the trial step again, until (2.18) is satisfied.

With the trial step s_k, we want to know if it produces enough reduction in the actual augmented Lagrangian relative to our approximate function. Thus, we compute the ratio

$$\rho_k \leftarrow \frac{\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_k + s_k, y_k, \mu_k)}{\Delta q_{\mathcal{L}}(s_k, x_k, y_k, \mu_k)}. \tag{2.19}$$

The success of reducing the actual augmented Lagrangian is determined with the help of constants $\eta_s \in (0, 1)$ and $\eta_{vs} \in (\eta_s, 1)$. If $\rho_k \geq \eta_s$, then the step is considered to be successful; and the trial step is accepted. If $\rho_k \geq \eta_{vs}$, then the step is considered to be very successful; we can be more aggressive with the trust region radius δ_k so it is expanded for the next iteration. On the other hand, if $\rho_k < \eta_s$, then the step does not produce enough reduction in the augmented Lagrangian; so the step is waved off and the trust region radius is reduced for the next iteration.

At last, the updated multiplier vector y_{k+1} needs to be defined. If the constraint violation at x_{k+1} is not sufficiently small compared to a tolerance value t_j, then y_{k+1} simply remains the same as y_k. If the constraint violation is sufficiently small, we check the condition

$$\|F_L(x_{k+1}, \pi(x_{k+1}, y_k, \mu_k))\|_2 \leq \|F_L(x_{k+1}, y_k)\|_2; \tag{2.20}$$

if it is satisfied, we set $\hat{y}_{k+1} \leftarrow \pi(x_{k+1}, y_k, \mu_k)$; otherwise, we set $\hat{y}_{k+1} \leftarrow y_k$. With $\hat{y}_{k+1}$ calculated, we then check if $\|F_L(x_{k+1}, \hat{y}_{k+1})\|$ or $\|F_{AL}(x_{k+1}, y_k, \mu_k)\|$ is sufficiently small compared to some tolerance value $T_j > 0$. If it is, new tolerance values $t_{j+1} < t_j$ and $T_{j+1} < T_j$ are calculated, and we set $y_{k+1} \leftarrow \hat{y}_{k+1}$. Otherwise, we set $y_{k+1} \leftarrow y_k$.

We conclude the description of Algorithm 4 by noting that (2.2) and (2.6) are theoretical conditions, and that in practical implementations positive tolerances should be used in the place of 0 for both. It should also be noted that for convergence guarantees, the algorithm in [2] controls the magnitudes of the computed multiplier estimates, but we do not include the procedure here as it is not included in our implementation.

Chapter 3

An Active Set Projected CG Method

In lines 14 and 17 of Algorithm 4, we need to solve the two trust-region subproblems to get the trial step s_k and the steering step r_k. The computation of the two is central to the adaptive Augmented Lagrangian trust region method we discussed previously, and also constitutes most of the algorithm's computational costs. In this chapter, we present a variant of the projected CG method to compute the exact, or at least a good approximate solution of the trust region subproblem.

As mentioned before, if we treat the trust region as an infinity norm constraint, the subproblems are just bound-constrained QPs. The projected CG method allows active set estimates to change rapidly between iterations and is often efficient when the constraints have such a simple form. That is the reason we chose it for our trust-region subproblems. For future reference, a general bound-constrained QP has the form

$$\min_x \; q(x) = \frac{1}{2} x^T G x + x^T c \tag{3.1a}$$

$$\text{subject to} \;\; l \leq x \leq u \tag{3.1b}$$

where G is symmetric and $\{l, u\}$ are vectors of lower and upper bounds of x, respectively.

A basic projected CG method involves cheap calculations and has convergence guarantees, but it may be inefficient, especially when it is unable to quickly identify the optimal active set. Thus, we combine the projected CG method with an active set update for an active set projected

CG method to get the exact, or at least a good approximate solution. In the rest of this chapter, we will describe the projected CG method and the active set estimation strategy in detail.

3.1 The Projected CG Method

The projected CG method consists of two stages. In the first stage, we search along the steepest descent direction from the origin point, and choose the Cauchy step s^C [5] as an initial estimate solution of the original problem. In the second stage, we estimate the active set at the optimal solution of the QP x^* and use a projected CG method [3, 7] to compute a solution of the original problem. We will discuss estimation of the active set in detail later.

3.1.1 First Stage: Cauchy Point Computation

The Cauchy step s^C is obtained by projecting the steepest descent direction onto the feasible region. Recall the projection operator P in (2.4), starting from the origin point, the Cauchy direction can be written as

$$s^C(\alpha) := P[-\alpha \nabla q(0)], \tag{3.2}$$

where $\alpha \geq 0$ is the step size.

The step size α is chosen so that the Cauchy step $s^C(\alpha)$ produces a sufficient reduction. In this case, we require that

$$q(s^C(\alpha)) \leq \mu_{pg} \nabla q(0)^T s^C(\alpha) \tag{3.3}$$

must be satisfied with some constant $\mu_{pg} \in (0, \frac{1}{2})$.

An iterative scheme is used to guarantee the Cauchy point $s^C(\alpha)$ is chosen in a finite number of evaluations. Given $\alpha^{(0)}$, we will generate a sequence of step sizes of the form

$$\alpha^{(k+1)} = \beta \alpha^{(k)},$$

where we either have $\beta > 1$ or $\beta < 1$ so the sequence is increasing or decreasing, respectively. The value of β is determined by $\alpha^{(0)}$. If the initial step size $\alpha^{(0)}$ fails to satisfy condition (3.3), the sufficient reduction condition, we choose $\beta < 1$ so the trial step size decreases until (3.3) is

satisfied. If the initial step size $\alpha^{(0)}$ satisfies (3.3), we choose $\beta > 1$ so the trial step size increase until it fails to satisfy (3.3). We then set the step size as the last one that satisfies (3.3). We then calculate the Cauchy step s^C with (3.2). The complete procedure is shown in Algorithm 5.

Algorithm 5 Cauchy step computation for Projected CG method

Choose constant $\beta_1 \in (0,1)$, $\beta_2 > 1$ and $\mu_{pg} \in (0,1)$.
Set $\alpha \leftarrow 1$ and compute $s^C(\alpha)$ with (3.2).
if (3.3) is satisfied, **then**
    while (3.3) is satisfied, **do**
        Set $\alpha_{temp} \leftarrow \alpha$, and then $\alpha \leftarrow \beta_2 \alpha$.
        Compute $s^C(\alpha)$ with (3.2).
    Set $\alpha \leftarrow \alpha_{temp}$.
else
    while (3.3) is not satisfied, **do**
        Set $\alpha \leftarrow \beta_1 \alpha$.
        Compute $s^C(\alpha)$ with (3.2).
return : $s^C(\alpha)$

3.1.2 Second Stage: Subspace Minimization

First, we define the working set $\mathcal{W}$ to be the set of bound constraints that are active at x. Once we have an estimation $\mathcal{W}$ of the optimal active set $\mathcal{A}(x^*)$ and x_0 as a starting point (the Cauchy step s^C gives the initial estimation $\mathcal{W}$ and x_0), we try to convert the problem to a equality-constrained QP which can be solved by many efficient algorithms, including the one we will be using for this stage: the projected CG method. We leave the discussion of estimating the active set for later, and focus on solving the QP with a working set $\mathcal{W}$ for now.

We modify the original problem by fixing the value of components x_0^i where $i \in \mathcal{W}$, and try to compute the remaining components of x from the subproblem (superscripts indicating components of a vector)

$$\min_x \; q(x) = \frac{1}{2} x^T G x + x^T c \tag{3.4a}$$

$$\text{subject to} \;\; x^i = x_0^i, \; i \in \mathcal{W}_k, \tag{3.4b}$$

$$l^i \leq x^i \leq u^i, \; i \notin \mathcal{W}_k. \tag{3.4c}$$

If the working set $\mathcal{W}$ is the same as $\mathcal{A}(x^*)$, solving (3.4) exactly is equivalent to solving (3.1)

exactly, and at the solution, only the equality constraints will be active. Notice that in this case the problem now can be regarded as an equality-constrained QP with (3.4a) and (3.4b), which can be rewritten in the general form

$$\min_x \ \frac{1}{2}x^T G x + x^T c$$
$$\text{subject to} \ \ A^T x = b.$$

Here, b would be a vector of all fixed values of $x_0^i, i \in \mathcal{W}$, and A^T is the matrix of constraint gradients. Note that problem of this form can be solved by a projected CG method, the solution of which, in the case, will also be the solution for (3.1).

However, even if the working set is different from the optimal active set, we can still use the projected CG to solve (3.4a) and (3.4b), to obtain an approximate solution that will help us find the exact solution. Algorithm 6 shows the procedure of the projected CG method, but we need to first show how projections are handled in the algorithm.

Algorithm 6 Projected CG Method

Set the starting point x_0.
Compute $r_0 \leftarrow Gx_0 + c$, $g_0 \leftarrow M_{proj} r_0$, $d_0 \leftarrow -g_0$, and choose $\kappa_{tol} > 0$.
Set $K \leftarrow 0$.
loop
    if $d^T G d \leq 0$, **then**
        return : $x^s \leftarrow x_K + \theta d_K$ where $\theta > 0$ is the largest value for x^s to be feasible.
    Set $\alpha_K \leftarrow r_K^T g_K / d_K^T G d_K$.
    Set $x_{K+1} \leftarrow x_K + \alpha_K d_K$.
    if x_{K+1}^i, where $i \notin \mathcal{W}$, encounters a bound, **then**
        return : $x^s \leftarrow x_{K+1}$.
    Set $r_{K+1} \leftarrow r_K + \alpha_K G d_K$.
    Set $g_{K+1} \leftarrow P r_{K+1}$.
    if $g_{K+1}^T r_{K+1} \leq \kappa_{tol} (g_0^T r_0)^{\frac{1}{2}}$, **then**
        return : $x^s \leftarrow x_{K+1}$.
    Set $\beta_{K+1} \leftarrow (r_{K+1})^T g_{K+1} / r_K^T g_K$.
    Set $d_{K+1} \leftarrow -g_{K+1} + \beta_{K+1} d_K$.
    Set $K \leftarrow K + 1$.

Notice that because of the nature of bound constraints, A^T will have a very simple form. We

define the scaled $n \times n$ projection matrix M_{proj} as

$$M_{proj} = Z(Z^T Z)^{-1} Z^T = I - A(A^T A)^{-1} A^T,$$

where Z is a null-space basis of A^T. In this implementation, we calculate projections $g = M_{proj} r$ by solving an augmented system of the form

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} g \\ v \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}. \tag{3.5}$$

Because A^T has a very simple form, we can compute g to obtain an explicit result. Assume the current working set is $\mathcal{W}$, solving (3.5) shows that g can be expressed as

$$g^i = \begin{cases} 0 & i \in \mathcal{W}, \\ r^i & otherwise. \end{cases} \tag{3.6}$$

The explicit result means no linear system is needed to be solved, and we can compute g very efficiently.

Because we have to accommodate the constraints in (3.4c) for the cases where the working set is different from the optimal active set, and there is the possibility of indefiniteness in the quadratic model, we use modified stop tests that are different than standard trust-region CG. We (a) terminate when negative curvature appears, and find a solution along the direction of the current step; (b) terminate if a bound in (3.4c) is encountered; or (c) terminate if $g_K^T r_K$ is smaller than a prescribed tolerance. In particular, when negative curvature is detected, it is preferred to search along the direction of the current step until a bound is encountered to obtain the largest feasible step.

We end this section by noting that in line 9, if the algorithm stops because a bound is encountered, we don't consider x^s a solution. Instead we will expand the active set and try to run projected CG with a new working set and new starting point. The details of this strategy will be discussed in the next section.

3.2 Estimating the Active Set

For bound-constrained QPs, we define an active set at a feasible point x as

$$\mathcal{A}(x) := \{i \,|\, x^i = l^i \text{ or } x^i = u^i\}.$$

Here we use superscript to represent the components in a vector. The working set $\mathcal{W}$ at x is then defined as

$$\mathcal{W} \leftarrow \mathcal{A}(x). \tag{3.7}$$

In order to compute the optimal solution x^* of the bound-constrained QP with the projected CG method, the optimal active set $\mathcal{A}(x^*)$ is necessary. The Cauchy step s^C gives us the first working set

$$\mathcal{W} \leftarrow \mathcal{A}(s^C),$$

but it may not be the same with the optimal active set $\mathcal{A}(x^*)$. An iterative approach is applied, so that by adding or removing elements, it modifies the working set with each iteration and eventually reaches the optimal active set. We summarize this active set projected CG method in Algorithm 7, and present the detail of the iterative approach as follows.

First, we discuss how to add elements to the working set $\mathcal{W}$. During Algorithm 6, if a component of x_K violates its bounds, then the algorithm is terminated. Then, we define a new point $\tilde{x}^s$ as

$$\tilde{x}^s \leftarrow P[x^s]. \tag{3.8}$$

We effectively project the current point x^s to the bounds to obtain $\tilde{x}^s$ so that it remains feasible to the bound-constrained QP. Instead of using it as an approximate solution, we use $\tilde{x}^s$ in (3.7) to update the working set $\mathcal{W}$. By doing this, those components that violate their bounds in x_K will be added to the working set; then we regard $\tilde{x}^s$ as the new starting point for Algorithm 6, so along with the new working set, Algorithm 6 can be started again to repeat the process we just described.

We need to be able to remove elements from the current working set as well. If Algorithm 6 finds a solution x^s for (3.4) with a working set $\mathcal{W}$ that is different from $\mathcal{A}(x^*)$, the solution may

Algorithm 7 Active Set Projected CG Method

Use Algorithm 5 to find the Cauchy point s^C.
Set $k \leftarrow 0$.
Set starting point $x_0 \leftarrow s^C$, and initialize working set $\mathcal{W} \leftarrow \mathcal{A}(x)$.
loop
    if number of elements in $\mathcal{W}$ is the same as x_0, **then**
        Use Algorithm 6 to compute x^s.
        while Algorithm 6 terminates because a bound is encountered, **do**
            Define $\tilde{x}^s$ by (3.8).
            Update the working set $\mathcal{W}$ by (3.7) with $\tilde{x}^s$, and set the new starting point $x_0 \leftarrow \tilde{x}^s$.
            Use Algorithm 6 to compute x^s.
    else
        Set $x^s \leftarrow x_0$.
    Compute the Lagrange multipliers z_l and z_u at x^s.
    if $z_l^i < 0$ or $z_u^i < 0$ holds for some $i \in \mathcal{W}$, **then**
        Update $\mathcal{W}$ by removing the element corresponding to the most negative multiplier from the working set.
    else
        return the first-order optimal solution $x^* \leftarrow x^s$.
    Set the new starting point $x_0 \leftarrow x^s$.

not satisfy the KKT conditions for the original problem (3.1), which include

$$Gx + c - z_l + z_u = 0 \tag{3.9a}$$

$$l \leq x \leq u \tag{3.9b}$$

$$z_l, z_u \leq 0 \tag{3.9c}$$

$$z_l^T(l - x) = 0 \tag{3.9d}$$

$$z_u^T(x - u) = 0. \tag{3.9e}$$

The quantities z_l and z_u are the vectors of Lagrange multipliers corresponding to the lower bound constraints and upper bound constraints respectively. For a solution x^s, (3.9b) is satisfied. Assuming (3.9a), (3.9d) and (3.9e) are satisfied as well, we can compute both z_l and z_u. More specifically, we must have $\{z_l^i, z_u^i\} = 0$ where $i \notin \mathcal{W}$, because those Lagrange multipliers correspond to inactive bounds. So whether or not the KKT conditions are satisfied only depends on $\{z_l^i, z_u^i\}$ where $i \in \mathcal{W}$. If $\{z_l^i, z_u^i\} \geq 0$ for all $i \in \mathcal{W}$, then (3.9c) is satisfied. That means the KKT conditions are satisfied. In this case, we have $\mathcal{W} = \mathcal{A}(x^*)$ and the exact solution for the original

problem is x^s.

If, on the other hand, $z_l^i < 0$ or $z_u^i < 0$ holds for some $i \in \mathcal{W}$, then (3.9c) is violated. We deal with this violation by removing one element i, corresponding to i such that $z_l^i < 0$ or $z_u^i < 0$, from the working set $\mathcal{W}$ and start Algorithm 6 from the current point x^s with the new working set. According to [6], the rate of decrease in the objective function when one constraint is removed is proportional to the magnitude of the Lagrange multiplier for that constraint. So we choose to remove the element corresponding to the most negative multiplier from the working set.

Notice in line 5 the condition requires that Algorithm 6 will be used only when the working set does not have the same number of elements as in x_0. Otherwise, we can see from (3.6) that no search direction from x_0 can be found for Algorithm 6. When this happens, we simply skip Algorithm 6 and check the KKT conditions at this point.

Chapter 4

Numerical Experiments

In this chapter we describe in detail the implementation of Algorithm 7 in MATLAB, which will be referred to as `ASPCG` from now on, as a subproblem algorithm for the adaptive AL algorithm, and provide some results from numerical experiments. In Section 4.2, we compare the performance of `ASPCG` and CPLEX's QP solver on a subset of the inequality constrained CUTEr [4] test problems. In Section 4.3 and Section 4.4, we explore and examine the performance of `ASPCG` when different values of certain parameters or different conditions are employed in the implementation of Algorithm 4.

4.1 Implementation Details

We already have the implementation of Algorithm 4 and a trust-region variant of Algorithm 1 by Curtis, Jiang and Robinson [2] in MATLAB, which will be referred to by `AAL` and `BAL` respectively, hereinafter. For the most part we will keep the implementations unchanged, and focus only on one critical component for both implementations: the algorithm for solving the trust region subproblem (2.14), and in the case of `AAL`, subproblem (2.10) as well. To be more specific, we focus on the part of the implementation for solving the subproblems in the case of inequality constrained problems, where our active set projected CG method can be implemented.

The original implementations of `AAL` and `BAL` use CPLEX's QP solver to solve the subproblems. The trust-region constraint of each subproblem is converted into an infinite norm trust-region constraint, and then the bound-constrained QPs are solved by CPLEX. Since the conversion of

the subproblems is necessary for ASPCG as well, we believe it is fair to compare the performance of the two.

There is one thing that is different between using CPLEX's QP solver and ASPCG. Since CPLEX requires the QP to be convex, the original implementations AAL and BAL add multiples of ten times the identity matrix to the Hessian of the QP when negative curvature is detected, until CPLEX can successfully return a solution. The algorithm we introduced in Chapter 3 does not need this procedure, so ASPCG differs slightly from Algorithm 6 and Algorithm 7 in dealing with negative curvature. If negative curvature occurs, ASPCG will simply terminate and use the current step x_K as the solution. The motivation for this implementation is to avoid unnecessary practical complications. For example, in some cases, the trust-region of a subproblem can become so big that expanding the current step x_K until a bound is encountered may be counterproductive.

As previously noted, the termination conditions for AAL and BAL should be practical variants of the theoretical ones. Both algorithms will terminate with a message of "optimal solution found" when

$$\|F_L(x_k, y_k)\|_\infty \leq \kappa_{opt} \text{ and } \|c_k\|_\infty \leq \kappa_{fea}, \tag{4.1}$$

and terminate with an infeasible stationary point when

$$\|F_{Feas}\|_\infty \leq 10^{-1}\kappa_{opt}, \|c_k\|_\infty > \kappa_{fea}, \text{ and } \mu_k < \mu_{min}. \tag{4.2}$$

Note that the implementation would only return an infeasible stationary point when the penalty parameter is small enough. In addition, we also set the algorithms to terminate if (4.1) and (4.2) are not satisfied within an iteration limit k_{max} and a CPU time limit t_{max}.

Table 4.1 and Table 4.2 below summarize the input parameter values that were chosen in algorithms BAL, AAL and ACPG.

Par.	Val.	Par.	Val.	Par.	Val.	Par.	Val.
β_1	5e-01	β_2	2e+00	μ_{pg}	1e-04	κ_{tol}	1e-08

Table 4.1: Parameter values used in ASPCG

Note that the implementation we use does not change the condition in line 10 of Algorithm 4 to make it more practical. The purpose of this condition has been discussed in Section 2.2; but

Par.	Val.	Par.	Val.	Par.	Val.	Par.	Val.
γ_μ	5e-01	κ_3	1e-04	Γ_δ	6e-01	κ_{opt}	1e-05
γ_t	5e-01	κ_t	9e-01	μ_0	1e-01	κ_{fea}	1e-07
γ_T	5e-01	η_s	1e-02	t_1	1e+00	μ_{min}	1e-20
γ_δ	5e-01	η_{vs}	9e-01	T_1	1e+00	κ_{max}	2e+03
κ_F	9e-01	δ	1e+04	δ_0	1e+00	t_{max}	1.8e3
κ_1	1e+00	κ_2	1e+00	δ_R	1e-04	ϵ	5e-01

Table 4.2: Parameter values used in AAL and BAL

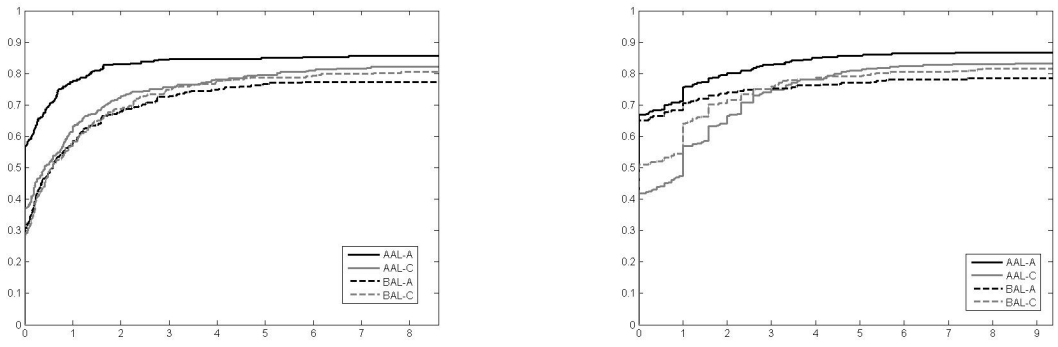
during initial numerical tests, we realized that a poorly chosen positive tolerance may drive down the value of the penalty parameter unnecessarily. So the condition remains unchanged in the implementation, so it is only satisfied when no feasible descent directions for $\mathcal{L}(\cdot, y_k, \mu_k)$ from x_k exist.

4.2 Comparison with CPLEX's QP Solver

In this section we observe the difference in performance between ASPCG and CPLEX's QP solver, when they are used as the subproblem solver in AAL and BAL. For this purpose, we choose a subset of the CUTEr test problems. The subset is chosen in the following way. First, the subproblem solver will only be used in the case of an inequality-constrained problem, so we eliminated all other types of problems. Second, we eliminated all problems with more than 1000 variables, as they become too large for the implementations in MATLAB to handle. Third, we eliminated problems that can't be solved by both AAL and BAL with either subproblem solver. This leaves us with a subset of 353 problems.

In order to fully examine the difference in performance, we will use both AAL and BAL with both ASPCG and CPLEX's QP solver for subproblems on the test problems. So we will be testing a total of 4 different implementations. We will call AAL with ASPCG as AAL-A, AAL with CPLEX as AAL-C, BAL with ACPG as BAL-A, and BAL with CPLEX as BAL-C hereinafter.

To measure the performance, we use performance profiles introduced by Dolan and Moré [1]. A performance profile uses a relative metric for judging performances, such as the number of iterations, and plots the fraction of problems solved by an algorithm within a multiple of the best algorithm according to said metric. Generally speaking, if we use the number of iterations as



(a) Iterations (b) CPU time

Figure 4.1: Performance profiles comparing `AAL-A`, `AAL-C`, `BAL-A`, and `BAL-C`.

the metric, a plot line that is on top towards the left side means the algorithm is more efficient, solving a better fraction of problems with fewer iterations; and a plot line that is on top towards the right side means the algorithm is more robust, solving more problems, regardless of how many iterations are taken.

The performance profiles in Figure 4.1 compare `AAL-A`, `AAL-C`, `BAL-A`, and `BAL-C` in terms of iterations and CPU time, respectively. These figures show several things. First and foremost, `AAL-A` is clearly superior to the rest in both efficiency and reliability on this collection of problems. Second, the adaptive AL algorithm implementations appear to have better reliability than the basic AL implementations; efficiency is harder to judge, although efficiency is better in `AAL-A` than `BAL-A`. For `AAL-C` and `BAL-C`, the adaptive AL method is more efficient in terms of iterations, but less efficient in terms of CPU time.

There are many reasons for an implementation to terminate, and Table 4.3 shows how many times each termination condition is triggered on the problems for each of the four implementations. Notice that the total number in each column does not add up to 353, which is the number

Cond.	`AAL-A`	`AAL-C`	`BAL-A`	`BAL-C`
Optimal solution found	306	294	277	288
Infeasible stationary point found	1	0	2	1
Reach iteration limit κ_{max}	33	50	60	47
Function evaluation error	5	2	6	3
Reach time limit t_{max}	8	1	8	10

Table 4.3: Termination condition tally comparing `AAL-A`, `AAL-C`, `BAL-A`, and `BAL-C`.

of problems that are in the test subset, for AAL-C and BAL-C; instead they are 347 and 349, respectively. This is because during the solution process of certain problems with the two implementations, some errors cause MATLAB to crush and thus no termination condition is triggered. These problems are considered unsolved in Figure 4.1.

The results in Table 4.3 appear to confirm that AAL-A has the best reliability. But it is necessary to check on the final value of the penalty parameter for the implementations before coming to this conclusion. If the penalty parameter is too small after the computation of a solution, the problem essentially becomes a problem of minimizing the constraint violation and will regard any feasible point as the optimal solution. Thus, it would be a good feature of an algorithm if μ_{final} does not go too small too often. We now present in Table 4.4 the number of final value μ_{final} of penalty parameter in each range for the four implementations.

μ_{final}	AAL-A	AAL-C	BAL-A	BAL-C
1e-01	129	128	161	154
[1e-02, 1e-01)	37	39	34	42
[1e-03, 1e-02)	35	46	40	28
[1e-04, 1e-03)	30	39	43	43
[1e-05, 1e-04)	44	35	36	54
[1e-06, 1e-05)	25	25	13	15
[1e-07, 1e-06)	16	14	7	9
(0, 1e-07)	37	21	13	4

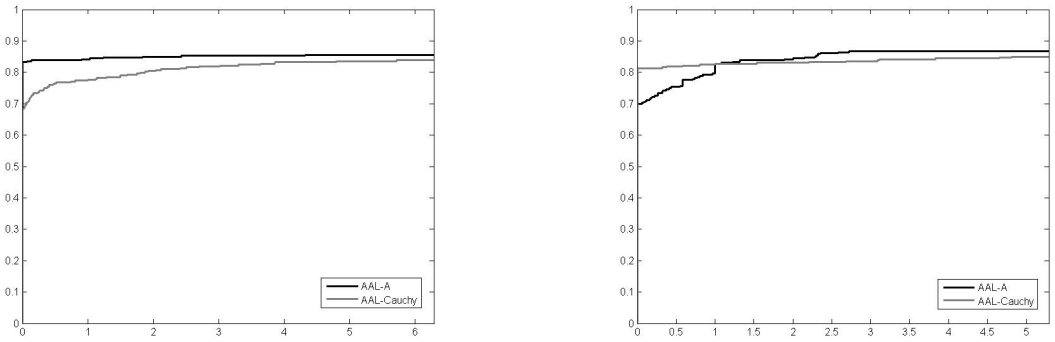
Table 4.4: Number of the final penalty parameter values are in the given ranges comparing AAL-A, AAL-C, BAL-A, and BAL-C.

We observe that the adaptive AL algorithm does cause the final penalty parameter value to be smaller than the basic AL algorithm. But it is not too small compared to the basic algorithm, so this could simply be because the adaptive strategy is working. We also note that AAL-A does not see too much drop in the final penalty parameter value compared to AAL-C, either. This shows that our subproblem algorithm has better efficiency and reliability than CPLEX's QP solver, without sacrificing too much on the final penalty parameter value on the test set of problems. This conclusion may be explained by looking at the strategy for updating the active set estimates. ASPCG uses a projected CG method to update the active set estimates. As we mentioned in Chapter 3, this allows the estimates to change rapidly between iterations, which may be an advantage over the active set method used in CPLEX, especially when the initial

estimate is not very accurate.

4.3 Different Steering Step

In this section and next, we look further into `AAL-A`, to observe the performance of our `ASPCG` method in different settings of `AAL`. The main difference between the adaptive AL approach and the basic one is the computation of a steering step before the trial step that tries to minimize the augmented Lagrangian. Because of the impact of the steering step on the value of the penalty parameter, we want to know the behavior of the AAL algorithm when the steering step is computed differently. In `AAL-A`, we use `ASPCG` for the computation of both the steering step and the trial step, and the implementation computes the exact or, at least, a very good approximate solution of both subproblems. We now modify `AAL-A` so the steering step is computed by an implementation of Algorithm 5, and the solution would be just the Cauchy step. This setup means that the steering step is now a less accurate approximate solution. We call the new implementation `AAL-Cauchy`. `AAL-A` and `AAL-Cauchy` were tested on the set of problems we used in Section 4.2.



(a) Iterations (b) CPU time

Figure 4.2: Performance profiles comparing `AAL-A` and `AAL-Cauchy`.

The performance profiles in Figure 4.2 compare `AAL-A` and `AAL-Cauchy` in terms of iterations and CPU time, respectively. The difference between the two lines is not very large. With iterations as the performance metric, `AAL-A` appears to be superior to `AAL-Cauchy` in both efficiency and reliability on this collection of problems. This may be a result of having less accurate steering steps which may lead to worse progress in some iterations. On the other hand, with CPU time

Cond.	AAL-A	AAL-Cauchy
Optimal solution found	306	300
Infeasible stationary point found	1	0
Reach iteration limit κ_{max}	33	42
Function evaluation error	5	5
Reach time limit t_{max}	8	6

Table 4.5: Termination condition tally comparing AAL-A and AAL-Cauchy.

μ_{final}	AAL-A	AAL-Cauchy	μ_{final}	AAL-A	AAL-Cauchy
1e-01	129	139	[1e-05, 1e-04)	44	41
[1e-02, 1e-01)	37	41	[1e-06, 1e-05)	25	25
[1e-03, 1e-02)	35	40	[1e-07, 1e-06)	16	18
[1e-04, 1e-03)	30	32	(0, 1e-07)	37	17

Table 4.6: Number of the final penalty parameter values are in the given ranges comparing AAL-A and AAL-Cauchy.

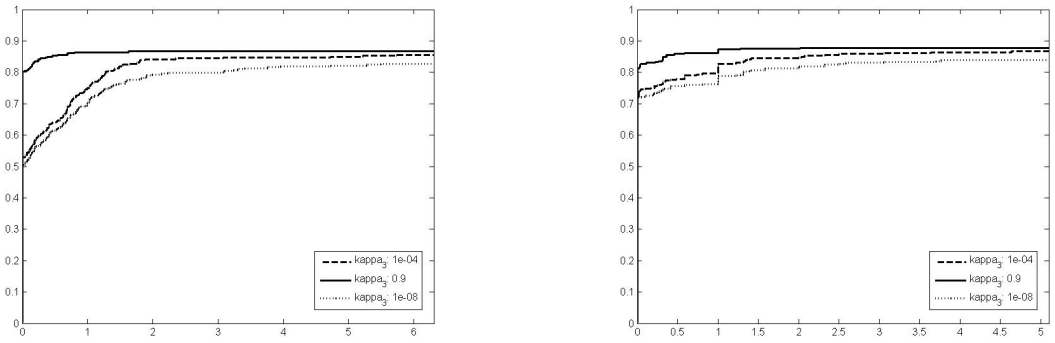
as the performance metric, AAL-Cauchy sees better efficiency than AAL-A. This may be because the computational costs of computing the Cauchy point is much less than solving the subproblem more accurately with ASPCG.

Similar to the previous section, we present in Table 4.5 how many times each termination condition is triggered on the collection of problems for both implementations; and Table 4.6 shows the number of final value μ_{final} of penalty parameter in each range for both implementations. We can confirm what we learn from Figure 4.2, in Table 4.5, that AAL-A and AAL-Cauchy solves similar numbers of problems, although AAL-A solves slightly more. But Table 4.6 suggests that AAL-Cauchy records more μ_{final} in the range [1e-04, 1e-01] than AAL-A, which, as we discussed in the last section, is a better distribution of μ_{final}. We expected these results, since AAL-A has more accurate steering steps, which, with condition (2.18c) implemented, drives the value of the penalty parameter down more aggressively. Note that AAL-A clearly records more μ_{final} in (0, 1e-07) than AAL-Cauchy. This may mean that AAL-A solves more problems than AAL-Cauchy, simply because it drives penalty parameter small in more cases, and it is only a feasible point that the implementation finds. Thus, we can't conclude that AAL-A shows better reliability, or efficiency, even if the performance profiles suggest so.

4.4 Different Value of the Parameter κ_3

We know that condition (2.18c) ensures that the trial step yields sufficient reduction in constraint violation compared to the steering step. In the last section, we observed the results of having inaccurate steering steps. In this section, we change the parameter κ_3 in (2.18c) which determines what fraction of constraint violation reduction of the steering step in trial step is considered "sufficient". Table 4.2 shows that so far our implementations use $\kappa_3 = $ 1e-04. We set it to 0.9 and 1e-08, as one value is close to 1 and the other is close to 0, and test the modified `AAL-A` on the same test set of problems as in Section 4.2.

We present the performance profiles of `AAL-A` when κ_3 is set at 0.9, 1e-04 and 1e-08, respectively, in Figure 4.3. Notice that larger κ_3 shows better efficiency and reliability in the profiles. However, in Section 4.3, we have seen that performance profiles do not take the final value of penalty parameter into consideration, and the reliability and efficiency in profiles may be misleading. Compared to the original $\kappa_3 = $ 1e-04, if instead κ_3 is set at 0.9, then usually only those trial steps that produces large reduction in constraint violation are considered valid, and the penalty parameter may be driven down more aggressively. On the other hand, if we use $\kappa_3 =$ 1e-08, trial steps would be accepted more easily and the penalty parameter decrease would be less frequent. The flaw for the former case has been discussed, the solution we computed may be for a feasibility problem instead of the original optimization problem. The flaw for the latter case is that sometimes the penalty parameter is not decreasing quickly enough, and we may need more iterations and time to reach a solution.



(a) Iterations (b) CPU time

Figure 4.3: Performance profiles comparing different κ_3.

Cond.	$\kappa_3 = 0.9$	$\kappa_3 =$ 1e-04	$\kappa_3 =$ 1e-08
Optimal solution found	310	306	296
Infeasible stationary point found	3	1	0
Reach iteration limit κ_{max}	32	33	45
Function evaluation error	3	5	5
Reach time limit t_{max}	5	8	7

Table 4.7: Termination condition tally comparing different κ_3.

μ_{final}	$\kappa_3 = 0.9$	$\kappa_3 =$ 1e-04	$\kappa_3 =$ 1e-08
1e-01	107	129	133
[1e-02, 1e-01)	37	37	41
[1e-03, 1e-02)	25	35	35
[1e-04, 1e-03)	25	30	25
[1e-05, 1e-04)	60	44	41
[1e-06, 1e-05)	30	25	29
[1e-07, 1e-06)	16	16	15
(0, 1e-07)	53	37	33

Table 4.8: Number of the final penalty parameter values are in the given ranges comparing different κ_3.

So we present in Table 4.7 how many times each termination condition is triggered on the collection of problems for different κ_3; and Table 4.8 shows the number of final value μ_{final} of penalty parameter in each range for all cases. First, we can see from Table 4.7 that the results match what we discussed about the flaw of having smaller κ_3, as we can see more problems can't be solved in the iteration limit when κ_3 change from 0.9 to 1e-08. Second, we can see from Table 4.8 that when κ_3 is set at 0.9, more penalty parameter final values are in the range (0, 1e-04) than the other two cases. This means that more problems that `AAL-A` solved in this case are in question as we are not sure how many of those solutions are optimal or just feasible. Thus we can't conclude that setting $\kappa_3 = 0.9$ is better for performance of `AAL-A` than the other two. On the other hand, the distribution of final penalty parameter value does not differ much between settings $\kappa_3 =$ 1e-04 and $\kappa_3 =$ 1e-08. So it is likely that $\kappa_3 =$ 1e-04 is a better setting for the performance of `AAL-A`, both in efficiency and reliability, than $\kappa_3 =$ 1e-08.

Chapter 5

Conclusion

We have presented, tested, and analyzed an algorithm for the subproblems of the AAL algorithm. It compute the exact, or at least a good approximate solution for the subproblems by using a projected CG method that estimates the optimal active set. With the subproblems being bound-constrained in form, the active set projected CG method we proposed can change the active set estimates rapidly between iterations; it makes finding the optimal estimate, and then a solution, quickly, even when the initial active set estimate is far from the optimal one. We have demonstrated that it outperforms the original subproblem solver implemented in the AAL algorithm in terms of both iterations and CPU running time over a wide range of problems while maintaining similar penalty parameter updates. We also show that the AAL algorithm with our method for solving subproblems is superior to the basic AL algorithm with the same subproblem solver in terms of iterations and CPU time on those problems.

A potential disadvantage of the AAL algorithm we used is the possibility that the penalty parameter will be driven too small, even when it is applied to solve problems where a constraint qualification is satisfied at all solution points. We examined this possibility by testing different steering steps and values of the parameter κ_3 in the AAL algorithm implementation with our subproblem solver. On the test problems, we showed that a rigorous requirement on the reduction of constraint violation in the trial step would drive the penalty parameter value down, and a loose requirement would have the opposite effect. Further inspection is necessary to determine how many of the solutions found in the implementations we tested are actually optimal, or just feasible

points. So we are unable to decide what the best constraint violation reduction requirement in the implementation would be. For now we only point out that smaller penalty parameter values are more likely to cause the returned solution to be just a feasible point.

In terms of the implementation of the active set projected CG method, one issue we have to take into consideration is numerical error in estimating the optimal active set. The theoretical conditions, which we used in our implementation of the subproblem solver, may cause the estimate to be very inaccurate in some cases. We didn't find this to be a persistent issue in our tests, but it is an important matter to consider in general.

Bibliography

[1] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with cops. Technical Report Technical Memorandum ANL/MCS-TM-246, Argonne National Laboratory, Argonne, IL, 2000.

[2] Frank E. Curtis, Hao Jiang, and Daniel P. Robinson. An adaptive augmented lagrangian method for large-scale constrained optimization. Technical Report 12T-016, COR@L Laboratory, Department of ISE, Lehigh University, 2013.

[3] Richard H. Byrd, Mary E. Hribar, and Jorge Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[4] Nicolas I. M. Gould, D. Orban, and Philippe L. Toint. Cuter and sifdec: A constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Software*, 29:373–394, 2003.

[5] Chin-Jen Lin and J. Moré. Newton's method for large bound-constrained optimizaion problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[6] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*, chapter 16, pages 467–480. Springer, second edition, 2006.

[7] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*, chapter 16, pages 461–463. Springer, second edition, 2006.

Biography

Wenda Zhang, born on February 20th, 1989 in Taiyuan, China, graduated in 2012 from Shanghai Jiao Tong University in China as an industrial engineering major. He then entered Lehigh University that fall, and is there now pursuing a Master of Science degree in the industrial and systems engineering program.