CARNEGIE MELLON UNIVERSITY

Advances in Newton-based Barrier Methods for Nonlinear Programming

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree of

DOCTOR OF PHILOSOPHY

in

CHEMICAL ENGINEERING

by

WEI WAN

MASTER OF COMPUTATIONAL DATA SCIENCE, CARNEGIE MELLON UNIVERSITY

B.S., CONTROL SCIENCE AND ENGINEERING, ZHEJIANG UNIVERSITY

Pittsburgh, Pennsylvania

Aug, 2017

Acknowledgments

I would first like to thank my adviser Larry Biegler for everything he done for me in the past five years. I feel so lucky I had the chance to work with Larry. He is patient, knowledgeable and full of ideas. Thanks for being such a wonderful adviser and thanks for being supportive and letting me explore and learn many things I am curious about.

I would also like to thank the members of my committee, Professors Ignacio Grossmann, Nick Sahinidis, and Javier Peña for all of their help, guidance, and encouragement during my PhD studies. I'd also like to thank all the members of Prof. Biegler's research group for all the discussions, conversation and encouragement. Lastly, I would like to thank my undergraduate adviser Xi Chen from Zhejiang University for leading me into PSE and encouraging and preparing me for graduate school.

I would like to gratefully acknowledge funding from the CAPD which supported this work. Additionally, I would like to thank the Masters of Computational Data Science program in the Language Technology Institute for providing me a strong master's education in Computer Science, which is indispensable for this work and my future career.

I would like to thanks all of my friends for the fun time we had in the these five years. Bethany Nicholson for all puzzle parties, Nick Austin, Clara Heuberger and Jens Bremer for taking me to a bar for my first time. Yash Puranik and Jess Cheng for teaching me how to make India Chai. Markus and Svenja Drouven for suggesting us to get married. Xue Yang for helping me settle down in Pittsburgh. Jun Shi and Mingzhao Yu for all the conversation and personal encouragement. Zhi Qian and Haoshui Yu for helping create a Chinese language environment for John. Yajun Wang and Zixi Zhao for staying in Pittsburgh with me for the whole five years. I would also like to thank all friends from the Master and PhD programs in ChemE and MCDS. All of you have made Pittsburgh such a wonderful place to live.

I'd like to thank my parents for their unconditional love and support for any of my choices. Finally, I would like to thank my husband John for his input for this work in discussions and proofreading, but most of all for his support, love, and constant encouragement. This work would not have been possible without you. Thanks for being the best friend, lab-mate and partner for the past five years and all the years to come.

Wei Wan
Pittsburgh, PA
Aug 2017

Abstract

Nonlinear programming is a very important tool for optimizing many systems in science and engineering. The interior point solver IPOPT has become one of the most popular solvers for NLP because of its high performance. However, certain types of problems are still challenging for IPOPT. This dissertation considers three improvements or extensions to IPOPT to improve performance on several practical classes of problems.

Compared to active set solvers that treat inequalities by identifying active constraints and transforming to equalities, the interior point method is less robust in the presence of degenerate constraints. Interior point methods require certain regularity conditions on the constraint set for the solution path to exist. Dependent constraints commonly appear in applications such as chemical process models and violate the regularity conditions. The interior point solver IPOPT introduces regularization terms to attempt to correct this, but in some cases the required regularization terms either too large or too small and the solver will fail. To deal with these challenges, we present a new structured regularization algorithm, which is able to numerically delete dependent equalities in the KKT matrix. Numerical experiments on hundreds of modified example problems show the effectiveness of this approach with average reduction of more than 50% of the iterations.

In some contexts such as online optimization, very fast solutions of an NLP are very important. To improve the performance of IPOPT, it is best to take advantage of problem structure. Dynamic optimization problems are often called online in a control or state-estimation. These problems are very large and have a particular sparse structure. This work investigates the use of parallelization to speed up the NLP solution. Because the KKT factorization is the most expensive step in IPOPT, this is the most important step to parallelize. Several cyclic reduction algorithms are compared for their performance on generic test matrices as well as matrices of the form found in dynamic optimization. The results show that for very large problems, the KKT matrix factorization time can be improved by a factor of four when using eight processors.

Mathematical programs with complementarity constraints (MPCCs) are another challenging class of problems for IPOPT. Several algorithmic modifications are examined to specially handle the difficult complementarity constraints. First, two automatic penalty adjustment approaches are implemented and compared. Next, the use of our structured regularization is tested in combination with the equality reformulation of MPCCs. Then, we propose an altered equality reformulation of MPCCs which effectively removes the degenerate equality or inequality constraints. Using the MacMPEC test library and two applications, we compare the efficiency of our approaches to previous NLP reformulation strategies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter we describe the background and motivation for this dissertation. The focus of the thesis is on solving problems in nonlinear programming with the solver IPOPT. Through solving these problems, several modifications to IPOPT are proposed. This chapter begins with an overview of the definition and applications of nonlinear programming. Then, we discuss the research challenges that motivated this work, and the structure of the thesis is outlined in detail.

1.1 Background of optimization

Mathematical modeling is an essential tool in engineering. Chemical processes are now described very effectively by mathematical expressions, including differential and algebraic equations. Computational simulations that solve these models are very important to predict the behavior of a new plant design, analyze a modification to an existing plant, or to develop a new control or operating policy. Good mathematical models offer cheap and safe ways to innovate and improve efficiency and profitability.

Once engineers have a way to predict outcomes of a model, it becomes natural to ask how to get the best outcome. By changing parameters such as the operating conditions (flowrates, temperature) or by considering alternative equipment sizing, one can try to optimize several objectives. These could include maximizing the net present value of a new project, finding operating specifications to maximize the efficiency of an existing plant,

or finding a control strategy to minimize the deviation from the those desired operating specifications. In these problems, there are often many degrees of freedom and the complex relationships between variables can make the optimal decisions non-intuitive. By using optimization algorithms in combination with mathematical process models, one can find practical solutions to engineering problems and often gain a better understanding of the underlying process.

Optimization problems are categorized by the type of objective under consideration and the mathematical characteristics of the model. Many process variables are continuous, such as temperature, flowrates, and pressures. If any of the variables are discrete, such as whether a certain piece of equipment will be built or not, then the problem is considered using mixed integer programming techniques. In this thesis, we will only consider optimization using continuous variables. Another important categorization is whether the model equations are linear or nonlinear. Chemical process models are usually nonlinear to consider the detailed thermodynamic and chemical behavior in the system. In addition, many variables naturally have bounds, for example pressure must be positive, but below a certain safety threshold.

The category of optimization problem considered in this dissertation is called nonlinear programming (NLP). These problems are nonlinear, usually nonconvex, with continuous variables and smooth constraints. This covers many important applications in chemical engineering including process optimization and optimal control. An NLP can be written mathematically as

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ s.t. \quad & c(x) = 0 \\ & x_L \leq x \leq x_U \end{aligned} \tag{1.1}$$

The objective function is $f : \mathbb{R}^n \to \mathbb{R}$ and equality constraints are $c : \mathbb{R}^n \to \mathbb{R}^m$ with

$m < n$. These functions are assumed to be sufficiently smooth. The upper and lower bounds x_L and x_U may be infinite for certain components, meaning that not every variable is required to have bounds. Note that form (1.1) is not unique, many NLP formulations include general nonlinear inequality constraints $g(x) \leq 0$. This can be easily transformed into form (1.1) through the addition of slack variables, for example $g(x)+s = 0$, with $s \geq 0$.

Many algorithms exist for solving nonlinear programs of form (1.1). Most attention in modern NLP solvers is given to solving large problems, possibly with hundreds of thousands of variables and constraints. Problems get larger as the detail and accuracy of the model increases. However, most NLPs in chemical engineering applications are highly sparse, meaning that efficient linear algebra routines can be used.

The NLP solver IPOPT (Interior Point OPTimizer) was originally developed by Andreas Wächter and Lorenz T. Biegler at Carnegie Mellon for the solution of large-scale nonlinear programs. The solver uses a filter line search combined with the interior point strategy. Linear algebra is handled with interfaces to several popular linear solvers. The original algorithm was written in Fortran and released open-source through COIN-OR. In 2005, IPOPT was updated with a new C++ implementation by Andreas Wächter and Carl Laird. This dissertation will consider further updates to the C++ version of IPOPT.

1.2 Problem challenges

IPOPT is widely used in many application areas including chemical industry, robotics, and economics. It is especially suitable for problems with many degrees of freedom. Compared to the active methods, the interior point approach avoids the combinatorial complexity of active set methods. In addition, interior point methods allow for the use of highly efficient sparse linear algebra routines. The IPOPT algorithm uses a filter line search, which

offers more lenient step acceptance criteria and often faster convergence compared to merit function methods.

Despite how efficient IPOPT is, there are still challenges which are listed as follows,

- Dependent equality constraints violate regularity conditions and may cause IPOPT to fail. However, these are common in chemical process models, for example in material flow equations for multicomponent streams. When composition is unknown, the mass balance equations involve bilinear terms, which may lead to degenerate equality constraints. In addition, synthesis or design MINLP problems derive NLP relaxation subproblems with the input flow rate vanishing when the corresponding unit is deselected. In that case, bilinear terms lead to local dependence in the equations at optima, which violate the Linear Independence Constraints Qualification (LICQ). Although IPOPT already has a regularization feature to try to deal with these problems, performance for dependent problems is still behind competing active set methods.
- Dynamic optimization includes detailed dynamic models within an optimization framework. Model predictive control, state estimation, and parameter estimation are all common applications of dynamic optimization, which can lead to significant improvements in process efficiency, reliability, safety, and profitability. After discretization, dynamic optimization problems can form large-scale, sparse nonlinear programs, often with many degrees of freedom. So IPOPT is a very suitable choice of solver. However, for use in online control, very fast solutions are required. With ever increasing system size, discretization level, and model complexity, fast solutions to dynamic optimization remains a challenge.
- Mathematical programming with complementarity constraints (MPECs) can be used to model certain classes of discrete events, including disappearance of phases, flow

reversal, and safety valve operation. MPCCs can be reformulated to NLP and solved by NLP solvers. For large process models, this approach can be more efficient than mixed integer optimization. However, the regularity conditions do not hold for the NLP reformulation of an MPCC at any feasible point, which is a challenge for all NLP solvers.

1.3 Research Statement and Dissertation Overview

This thesis addresses three categories of challenging models. First, we proposed the structured regularization algorithms which can eliminate the dependent constraints locally. The elimination maintains the high performance of IPOPT though Newton steps and helps the convergence for models with globally and locally dependent constraints. The convergence of IPOPT with structured regularization is proved as well. The second challenge is related to dynamic optimization problems. Taking advantage of the special KKT linear system structure created by dynamic optimization problems after discretization, we apply two parallel cyclic reduction methods instead of the standard linear solver. Finally, for MPCCs, we test two auto adjustment penalty methods, based on the penalty reformulation that can adjust the penalty term automatically. We also propose a constraint elimination method to eliminate the dependent constraints for MPCCs.

This dissertation is organized as follows:

Chapter 2 starts with the basic concepts of nonlinear programming including the first and second order KKT conditions and constraint qualifications. These concepts give theoretical foundation for the numerical solution of NLPs. Then, we discuss the development of practical algorithms to solve NLPs. Finally, popular state-of-art solvers are introduced with the comparison of the strengths and weaknesses.

Chapter 3 provides an overview of the state-of-art NLP solver IPOPT, which implements a primal-dual interior-point algorithm with a filter line-search method for nonlinear programming. In this chapter, first we discuss the primal-dual barrier approach, then apply Newton steps to get the solution of the barrier problem. A line search filter method is used to guarantee the quality of the step and then a restoration phase is defined for the filter method. When introducing the algorithm, we use a simple format of NLP with only equalities and lower bounds to simplify the notation. However, we also discuss the practical issue of how IPOPT deals with inequality constraints and upper and lower bounds. Finally, we discuss both the global and local convergence proofs for IPOPT.

Chapter 4 introduces a new structured regularization strategy; within the Newton step it identifies an independent subset of equality constraints and removes the remaining constraints without modifying the KKT matrix structure. This approach leads to more accurate Newton steps and faster convergence, while maintaining global convergence properties. Implemented in IPOPT with linear solvers HSL MA57, HSL MA97 and MUMPS, we present numerical experiments on hundreds of examples from the CUTEr test set, modified for dependency. These results show an average reduction in iterations of more than 50% over the current version of IPOPT. In addition, several nonlinear blending problems are solved with the proposed algorithm, and improvements over existing regularization strategies are further demonstrated.

Chapter 5 explores a parallel decomposition strategy for block tridiagonal systems that is based on the cyclic reduction (CR) method applied to the KKT linear system of dynamic optimization problems. The classical CR method has good observed performance, but its numerical stability properties need further study for our KKT system. An alternative method proposed by Yalamov and Pavlov has better theoretical stability guarantees. We compare traditional CR to the Yalamov variant and discuss modifications to the CR de-

composition that improve performance our KKT systems. Finally we apply the approach to four industrially relevant dynamic optimization case studies. On the largest problem, a parallel speedup of a factor of four is observed when using eight processors.

Chapter 6 discusses our experiences in extending IPOPT to deal with the solution of MPCCs. First, we propose two automatic penalty adjustment approaches in IPOPT. The first approach is similar to that from [1], in which the complementary error is checked and the penalty parameter is potentially adjusted after each barrier problem is converged. The second approach only adjusts the penalty term after the penalty NLP converges. Next, we propose an improvement to the equality reformulation of MPCCs. When a complementary constraint is biactive, the equality constraint leads to a singular row in the Jacobian. We address this using the structured regularization strategy proposed in Chapter 4. In addition, we propose an altered equality reformulation of MPCCs that extends this concept to numerically remove the degenerate equality or inequality constraints. The MacMPEC test library is used to compare MPCC solution methods. We also demonstrate performance on two applications.

Chapter 7 concludes this dissertation, lists the major contributions and discusses recommendations for future work.

Chapter 2

NLP Background

This chapter will briefly review the basics of nonlinear programming theory and computation. First, the first-order KKT necessary conditions are reviewed, followed by a discussion of constraint qualifications. Then we present the second order necessary and sufficient optimality conditions. In the second part of this chapter, we give an overview of algorithms for nonlinear programming, from sequential unconstrained methods to SQP and interior point methods. Finally we compare several of the most popular modern NLP solvers.

2.1 Optimality Conditions

In this thesis, we will focus on gradient-based solution methods that guarantee convergence to a local optimum of a general NLP. For this section, we formulate the NLP in the following form for simplicity of notation, where bounds and inequalities are represented as $g_i(x) \leq 0$. This gives the following formulation:

$$\begin{aligned} \min \quad & f(x) \\ s.t. \quad & c(x) = 0 \\ & g(x) \leq 0 \end{aligned} \tag{2.1}$$

where we define decision variables $x \in \mathbb{R}^n$, objective function $f : \mathbb{R}^n \to \mathbb{R}$, equality constraints $c : \mathbb{R}^n \to \mathbb{R}^{n_e}$, and inequality constraints $g : \mathbb{R}^n \to \mathbb{R}^{n_i}$.

Sometimes it is useful to have a quick way to refer to the feasible set of (2.1). We define the set $\mathcal{X}$ as the set $x \in \mathbb{R}^n$ such that $c(x) = 0$ and $g(x) \leq 0$.

We use the definition of the feasible set $\mathcal{X}$ to define a local minimum of (2.1). A point x^* is a local minimum of (2.1) if $x^* \in \mathcal{X}$ and there exists $\epsilon > 0$ such that:

$$f(x) \geq f(x^*) \quad \forall\, x \in \mathcal{X} \cap \{\|x - x^*\| \leq \epsilon\}. \tag{2.2}$$

2.1.1 First Order KKT Conditions

The first order Karush-Kuhn-Tucker (KKT) conditions provide necessary conditions to characterize a local minimum x^*. The only additional requirement to use the KKT conditions is that some constraint qualification holds at the local solution x^*. Constraint qualifications are discussed in the following section.

To make the presentation of optimality conditions more clear, we first define the Lagrangian function as follows:

$$\mathcal{L}(x, \lambda, \eta) = f(x) + c(x)^T\lambda + g(x)^T\eta \tag{2.3}$$

The KKT necessary conditions are the result of the following theorem:

Theorem 1. *If x^* is a local minimum of (2.1) and a constraint qualification holds at x^*, then there exist multipliers λ^* and η^* such that*

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*, \eta^*) = \nabla f(x^*) + \nabla c(x^*)\lambda^* + \nabla g_j(x^*)\eta^* &= 0 \\ c(x^*) &= 0 \\ g(x^*) &\leq 0 \\ \eta^* &\geq 0 \\ g(x^*)^T\eta^* &= 0 \end{aligned} \tag{2.4}$$

2.1.2 Constraint Qualifications (CQs)

Constraint qualifications (or regularity conditions) contain information about the relationship between the feasible set and the set that results from local linearizations of the con-

straints. Without a constraint qualification, the optimal solution may or may not be a KKT point. Here, we consider two constraint qualifications most often used in nonlinear optimization.

For notation in this section, define $\mathcal{I} = \{1, \ldots, n_e\}$ and $\mathcal{J} = \{1, \ldots, n_i\}$ as the index sets for equality and inequality constraints respectively. The j^{th} inequality constraint is written $g_j(x)$.

Given a point x, we define the notion of an active set of inequalities $\mathcal{A}(x) \subseteq \mathcal{J}$ as follows:

$$\mathcal{A}(x) = \{j \mid g_j(x) = 0\} \tag{2.5}$$

Now we define the linear independence constraint qualification (LICQ) and Mangasarian-Fromovitz constraint qualification (MFCQ) as follows:

Definition (LICQ). Given a point x and corresponding active set $\mathcal{A}(x)$, LICQ is defined by linear independence of the constraint gradients $\{\nabla c_i(x), \nabla g_j(x)\}$ for all $i \in \mathcal{I}$ and $j \in \mathcal{A}(x^*)\}$.

Definition (MFCQ). Given a point x and corresponding active set $\mathcal{A}(x)$, MFCQ is defined by linear independence of the equality constraint gradients and the existence of a search direction p such that $\nabla c_i(x)^T p = 0$ for $i \in \mathcal{I}$ and $\nabla g_j(x)^T p < 0$ for all $j \in \mathcal{A}(x)$.

From the definitions, it is easy to show LICQ implies MFCQ, thus MFCQ is the weaker condition. If LICQ holds at a local optimal solution, the multipliers can be solved uniquely. However, if there are dependent constraints in the active set but MFCQ holds, all multipliers are bounded in a polytope [2].

There are other constraint qualifications that are valid for the KKT theorem, but most are not easily verified in practice. We will address a problem specific extension of CQs in Chapter 6.

2.1.3 Second Order Conditions

In addition to the first-order conditions, local minimizers may be further characterized using second order information. There are both second order necessary optimality conditions and second order sufficient optimality conditions. First define the following cone:

$$d \in \mathcal{C}(x, \lambda, \eta) \iff \begin{cases} \nabla c(x)^T d = 0, \\ \nabla g_j(x)^T d = 0, \quad j \in \{i \mid g_i(x) = 0, \lambda_i > 0\} \\ \nabla g_j(x)^T d \leq 0, \quad j \in \{i \mid g_i(x) = 0, \lambda_i = 0\} \end{cases} \tag{2.6}$$

Theorem 2. *If x^* is a local minimum of (2.1), LICQ holds at x^*, and λ^*, η^* are the multipliers that satisfy the KKT conditions* (2.4)*, then:*

$$d^T \nabla_{xx} \mathcal{L}(x^*, \lambda^*, \eta^*) d \geq 0 \quad \forall\, d \in \mathcal{C} \tag{2.7}$$

The second order sufficient conditions are given by the following theorem:

Theorem 3. *Suppose that x^* and multipliers λ^*, η^* satisfy the KKT conditions* (2.4) *and*

$$d^T \nabla_{xx} \mathcal{L}(x^*, \lambda^*, \eta^*) d > 0 \quad \textit{for all nonzero } d \in \mathcal{C} \tag{2.8}$$

then x^* is a strict local solution of (2.1)

For proofs of these theorems, see for example [3].

2.2 NLP Solution Methods

In this section we briefly review the history of numerical methods for solving nonlinear programming problems. The focus is on nonlinear, nonconvex problems with general constraints as shown in (2.1).

2.2.1 Older Methods

The earliest algorithms for nonlinear programming were usually based on the sequential unconstrained minimization technique. These methods took advantage of well-established unconstrained minimization techniques (usually employing Newton's method) to solve a sequence of subproblems. The constraints can be handled in one of two ways. The first method, known as penalty functions, creates a new objective function where constraint violation is penalized with a scalar $\rho \in \mathbb{R}^+$

$$f_{pen}(x, \rho) = f(x) + \rho\|c(x)\| + \rho\|\max(0, g(x))\| \tag{2.9}$$

It can be shown that there exists a sufficiently large penalty parameter ρ such that a local minimum x^* of (2.1) is also a local minimum of (2.9). Thus, by solving a sequence of unconstrained subproblems $\min_x f_{pen}(x, \rho_k)$ for an increasing sequence $\{\rho_k\}$, we can converge to a solution of (2.1).

By typically starting with a low penalty parameter ρ, the subproblem solutions will usually begin outside the feasible set, only reaching feasibility once a sufficiently large ρ has been found. An alternative approach is the barrier method, where iterates stay in the strict interior with respect to the inequality constraints during the optimization process. If the problem does not have equality constraints, the barrier problem is formed as follows:

$$f_{bar}(x, \mu) = f(x) - \mu \sum_{j \in \mathcal{J}} \log(-g_j(x)) \tag{2.10}$$

Handling equality constraints with barrier methods would require other modifications. Barrier methods have regained popularity more recently with the development of interior point methods, discussed below.

More recent nested minimization algorithms are often based on the augmented Lagrangian function. Consider for now an equality constrained NLP. The augmented La-

grangian is defined as:

$$L_A(x, \lambda, \rho) = f(x) + c(x)^T\lambda + (\rho/2)c(x)^T c(x) \tag{2.11}$$

If λ^* corresponds to the equality constraint multipliers at the solution, then $L_A(x, \lambda^*, \rho)$ is an exact penalty function for sufficiently large ρ. Compared to (2.9), this penalty function is smooth, so it leads to easier subproblem solutions. The multiplier estimates λ are normally generated in an outer loop, for example using the least squares estimate:

$$\lambda(x) = -[\nabla c(x)^T \nabla c(x)]^{-1} \nabla c(x)^T \nabla f(x) \tag{2.12}$$

The penalty parameter ρ also has to be updated iteratively in the outer iteration, and a suitable value is not easy to determine. Too large values may cause ill-conditioning, while too small do not guarantee finding a solution. In augmented Lagrangian methods, inequality constraints $g(x) \leq 0$ can be handled in several ways, for example by slacking and treating bounds explicitly in subproblem solutions or through use of a barrier term.

2.2.2 Sequential Quadratic Programming (SQP)

As the name implies, sequential quadratic programming SQP solves an NLP through a sequence of quadratic programming approximations. The motivation for this approach comes from the following observation. Assume that there are no inequality constraints in an NLP. Then the KKT conditions are

$$\begin{aligned} \nabla f(x^*) + A(x^*)\lambda^* &= 0 \\ c(x^*) &= 0 \end{aligned} \tag{2.13}$$

where $A(x) = \nabla c(x)$. If we solve this system of equations using Newton's method, then we will generate a sequence of points $\{x_k, \lambda_k\}$ converging to $\{x^*, \lambda^*\}$. Define the step from iteration to iteration as d_k^x and d_k^λ such that $x_{k+1} = x_k + d_k^x$ and $\lambda_{k+1} = \lambda_k + d_k^\lambda$. This

Newton step is generated by solving the following linear system:

$$\begin{bmatrix} W_k & A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k \lambda_k \\ c(x_k) \end{pmatrix} \tag{2.14}$$

where $W_k = \nabla_{xx} \mathcal{L}(x_k, \lambda_k)$ and $A_k = \nabla c(x_k)$ The interesting point is that solving this linear system is equivalent to finding a critical point of the quadratic program

$$\begin{aligned} \min \quad & \nabla f(x_k)^T d + \tfrac{1}{2} d^T W_k d \\ s.t. \quad & A_k^T d = 0 \end{aligned} \tag{2.15}$$

If W_k is positive definite on the null space of A_k^T, then d_k^x is the solution of QP (2.15). This observation forms the basis of SQP methods. The inequality constraints can be handled by an active set strategy in the QP, where active constraints are guessed and treated as equalities. It can be shown that under mild assumptions the linearized QP can identify the correct active set. See [4, 5, 6, 7] for more details on SQP and related active set methods.

2.2.3 Interior Point Methods

An alternative approach to handling inequality constraints is interior point methods. In interior point methods, the inequality constraints are handled with a barrier term, while the step generation still utilizes linearized constraints. It is useful to consider the NLP in the form

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t. } c(x) = 0, \;\; x \geq 0 \tag{2.16}$$

After moving bound constraints to the objective function, the barrier subproblem is formed as follows:

$$\min_{x \in \mathbb{R}^n} \varphi_\mu(x) = f(x) - \mu \sum_{i=1}^{n} \ln(x^{(i)}) \;\; \text{s.t. } c(x) = 0, \tag{2.17}$$

where $\mu \geq 0$ is the barrier parameter and $\varphi_\mu(x)$ is called the barrier function. The solutions of (2.17) converge to the solution of (2.16) as $\mu \to 0$ under certain conditions, see [2] or [8].

After introducing dual variables

$$v^{(i)} = \frac{\mu}{x^{(i)}}$$

the KKT conditions of (2.17) are equivalent to the *primal-dual equations*:

$$\begin{aligned} \nabla f(x) + A(x)\lambda - v &= 0 \\ c(x) &= 0 \\ XVe - \mu e &= 0 \end{aligned} \tag{2.18}$$

Interior point methods solve the primal dual equations in inner iterations, and control μ in an outer iteration. IPOPT is an interior point method, the detailed algorithm is introduced in Chapter 3.

2.3 Popular Solvers

In this section, popular NLP solvers are briefly reviewed. The solvers are divided into active set approaches and interior point approaches.

2.3.1 SQP solvers

1. **filterSQP** [9]: Filter SQP is a trust region SQP method, using the filter method for globalization. It uses the *bqpd* package to solve QPs. In the case of indefinite Hessians, a local QP solution is returned.

2. **KNITRO** [10]: The KNITRO package includes several solvers. Although better known for its interior point methods, KNITRO also implements an SLQP algorithm. An LP approximation of the ℓ_1 penalty function estimates an active set, then an equality constrained QP is solved. The search direction includes information from both subproblems.

3. **NLPQLP** An extension of the SQP solver NLPQL [11], NLPQLP implements a non-monotone line search. The Hessian is updated by the modified BFGS formula. Dense linear algebra is used, so it is more suitable for smaller problems.

4. **NPSOL** [12] is an SQP algorithm using a linear search on the augmented Lagrangian merit function. The Hessian is approximated with a dense BFGS update. Since linear algebra is dense, it is more suitable to smaller problems.

5. **SNOPT** [13]: Also developed by Gill, Murray, and Wright [14], SNOPT implements sparse linear algebra routines so it is suitable for larger problems. A full-space, limited-memory BFGS update is used for the Hessian matrix.

2.3.2 Interior Point

1. **IPOPT** [15]: IPOPT uses a filter line search method to solve the primal dual equations. It is available through the open-source COIN-OR project. The full algorithm of IPOPT is given in Chapter 3.

2. **KNITRO** [10]: KNITRO includes the interior point method described in [16]. It can switch between merit function line search and a trust region approach depending on the problem features. The KKT matrix can be solved with either a direct factorization or indirect conjugate gradient method.

3. **LOQO** [17]: LOQO uses a line search method combined with elements of a filter method, with recourse to a merit function in certain circumstances.

2.3.3 Nested and Gradient Projection

1. **CONOPT** [18]: CONOPT includes several active set NLP solvers. These solvers are chosen automatically and sometimes nested. These include a gradient projection

method, a sequential linear programming method, and an SQP-type method. Efficient sparse linear algebra is used.

2. **MINOS** [19]: MINOS is a reduced space augmented Lagrangian method. The augmented Lagrangian subproblem is solved with linearized constraints. A reduced gradient method with quasi-Newton updates is then used to solve the subproblem.

3. **LANCELOT** [20]: LANCELOT is an augmented Lagrangian method using a trust region approach. After moving equalities to the objective with the augmented Lagrangian function, the bound constrained trust region subproblem is solved with conjugate gradient steps.

4. **PENNON** [21]: PENNON first moves the constraints to the objective using a penalty/barrier function. The unconstrained augmented Lagrangian subproblems are solved with a Newton-type solver. The solver also supports semidefinite constraints on matrices.

Chapter 3

Review of IPOPT Algorithm

In the previous chapter, we introduced nonlinear programming through optimality conditions, algorithmic concepts, and an overview of software packages. In this chapter, we will focus on an interior point approach (IPOPT) and discuss the detailed implementation and convergence proof.

3.1 Background of IPOPT

3.1.1 Primal-dual barrier approach

We consider an NLP problem of the form,

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{s.t. } c(x) = 0, \;\; x \geq 0 \tag{3.1}$$

and assume that $f(x) : \mathbb{R}^n \to \mathbb{R}$ and $c(x) : \mathbb{R}^n \to \mathbb{R}^m$, with $m < n \in \mathbb{N}$, have continuous first and second derivatives. In IPOPT, the inequality constraints are substituted by a barrier function with barrier parameter $\mu \in \mathbb{R}^+$. Thus, we compute approximate solutions to problem (3.1) by solving a sequence of problems of the form

$$\min_{x\in\mathbb{R}^n} \varphi_\mu(x) = f(x) - \mu \sum_{i=1}^{n} \ln(x^{(i)}) \;\; \text{s.t. } c(x) = 0, \tag{3.2}$$

where $x^{(i)}$ denotes the i-th component of a vector $x \in \mathbb{R}^n$. By defining $v = \mu X^{-1} e$, a KKT point of problem (3.2) satisfies the *primal dual equations*:

$$\begin{aligned} \nabla f(x) + A(x)\lambda - v &= 0 \\ c(x) &= 0 \\ XVe - \mu e &= 0 \end{aligned} \tag{3.3}$$

where $e = (1, ..., 1) \in \mathbb{R}^n$ and $A(x) = \nabla c(x) \in \mathbb{R}^{n \times m}$. Here $\lambda \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ correspond to Lagrange multipliers for equality constraints and bound constraints, respectively. X and V are diagonal matrices where $X = diag(x)$ and $V = diag(v)$.

The primal dual equations are solved repeatedly with decreasing values of μ. For each fixed μ, we define the optimality error to check for convergence for the barrier problem as

$$E_\mu(x, \lambda, v) := \max \left\{ \frac{\|\nabla f(x) + A\lambda - v\|_\infty}{s_d}, \|c(x)\|_\infty, \frac{\|XVe - \mu e\|_\infty}{s_c} \right\} \tag{3.4}$$

with scaling parameters $s_d, s_c \geq 0$ defined as follows ($s_{max} \geq 1$ is a scaling threshold),

$$s_d = \max \left\{ s_{max}, \frac{\|\lambda\|_1 + \|v\|_1}{m + n} \right\} / s_{max} \quad s_c = \max \left\{ s_{max}, \frac{\|v\|_1}{n} \right\} / s_{max} \tag{3.5}$$

When the optimality error is below a certain tolerance, the barrier parameter is decreased. When the optimality error of equation (3.4) with $\mu = 0$ is below a tolerance ϵ_{tol}, the overall algorithm terminates and the optimal solution is found.

3.1.2 Solution of the barrier problem

Following the notation in [15], j denotes the iteration counter for the "outer loop" representing the sequence μ_j, and k denotes the iteration counter for the "inner loop" used to solve the primal dual equations (3.3) at fixed μ_j. In the inner loop we solve the primal-dual equations using Newton's method. The Newton step $\left(d_k^x, d_k^\lambda, d_k^v\right)$ in the k^{th} iteration

is calculated by,

$$\begin{bmatrix} W_k & A_k & -I \\ (A_k)^T & 0 & 0 \\ V_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \\ d_k^v \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k\lambda_k - v_k \\ c(x_k) \\ X_k V_k e - \mu_j e \end{pmatrix} \tag{3.6}$$

where W_k denotes the Hessian $\nabla^2_{xx}(f(x) + c(x)^T\lambda_k)$ at x_k and $A_k = A(x_k)$. Because of the barrier terms, $x_k > 0$ for all k and we can define $\Sigma_k = X_k^{-1}V_k$. Eliminating d_k^v in (3.6) leads to the symmetric matrix shown in the following linear system:

$$\begin{bmatrix} W_k + \Sigma_k & A_k \\ (A_k)^T & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla\varphi_{\mu_j}(x_k) + A_k\lambda_k \\ c(x_k) \end{pmatrix} \tag{3.7}$$

Note that d_k^v is required in the iterations and is calculated through $d_k^v = \mu X_k^{-1}e - v_k - \Sigma_k d_k^x$, explicitly.

To obtain well-defined Newton steps, we need to ensure the inertia of the KKT matrix in (3.7) is (n, m, 0), namely, n positive, m negative, and no zero eigenvalues. The satisfaction of the inertia condition implies that A_k is full column rank and the projection of $W_k + \Sigma_k$ onto the null space of the Jacobian matrix A_k^T is positive definite. Thus the Newton step can be solved uniquely. However, the inertia is not always correct during the iteration. IPOPT regularizes the KKT matrix with δ_x and δ_c, and solves the linear system

$$\begin{bmatrix} W_k + \Sigma_k + \delta_x I & A_k \\ (A_k)^T & -\delta_c I \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla\varphi_{\mu_j}(x_k) + A_k\lambda_k \\ c(x_k) \end{pmatrix} \tag{3.8}$$

instead. The algorithm to obtain δ_x and δ_c is introduced in section 3.1.4.

3.1.3 A Line search filter method

After obtaining a search direction d_k from equation (3.8), IPOPT applies a line-search filter method to obtain the step. The line search is started from $\alpha_{k,l} := min(\alpha_k^{max}, 1)$ to define

trial iterates $x_k(\alpha_{k,l}) := x_k + \alpha_{k,l} d_k$ where α_k^{max} is the largest value so that $x_k(\alpha_k^{max})$ is inside the bounds. The step size is decreased with the sequence $\alpha_{k,l} = 2^{-l}\alpha_k^{max}$ (with $l = 0, 1, 2...$) to try to meet the following acceptance criteria.

A two dimensional filter [9] of the form $\mathcal{F}_k := \{\theta(x), \varphi(x)\}$ with $\theta(x) = \|c(x)\|$ defines a prohibited region for ordered pairs (θ, φ). If a trial point $x_k(\alpha)$ is acceptable by the filter (i.e. $(\theta(x_k(\alpha_{k,l})), \varphi_{\mu_j}(x_k(\alpha_{k,l})))) \notin \mathcal{F}_k$), the following conditions are considered to check whether a trial iterate should be accepted.

- Switching Condition (SC): $-m_k(\alpha_{k,l}) > 0$ and $[-m_k(\alpha_{k,l})]^{s_\varphi}[\alpha_{k,l}]^{1-s_\varphi} > \kappa_\theta[\theta(x_k)]^{s_\theta}$
- Armijo Condition (AC): $\varphi_{\mu_j}(x_k(\alpha_{k,l})) \leq \varphi_{\mu_j}(x_k) + \eta_\varphi m_k(\alpha_{k,l})$
- Sufficient Decrease Condition (SDC): $\theta(x_k(\alpha_{k,l})) \leq (1-\gamma_\theta)\theta(x_k)$ or $\varphi_{\mu_j}(x_k(\alpha_{k,l})) \leq \varphi_{\mu_j}(x_k) - \gamma_\varphi\theta(x_k)$

Here $\kappa_\theta > 0$, $s_\theta > 1$, $s_\varphi \geq 1$ and $\gamma_\varphi, \gamma_\theta, \eta_\varphi \in (0, 1)$ are given constants and $m_k(\alpha) := \alpha\nabla\varphi_{\mu_j}(x_k)^T d_k$ is a linear model for the improvement of the barrier objective function.

There are two possible cases to accept the step. First, if SC and AC hold, we accept the step because it is a descent direction and has sufficient improvement of the objective function. This trial step size $\alpha_{k,l}$ is defined as an *f-step-size*. However, if SC doesn't hold, SDC is checked instead of AC as a standard for the progress in either the barrier objective function or the primal infeasibility $\theta(x)$. If SDC holds, we accept the trial point as well.

If a trial iterate is accepted as an f-step-iteration, the filter remains unchanged ($\mathcal{F}_{k+1} = \mathcal{F}_k$). Otherwise, the filter is augmented for the next iteration using the formula

$$\mathcal{F}_{k+1} := \mathcal{F}_k \cup \{(\theta, \varphi) | \theta \geq (1-\gamma_\theta)\theta(x_k), \varphi \geq \varphi(x_k) - \gamma_\varphi\theta(x_k)\} \tag{3.9}$$

Finally, for some cases it is not possible to find a trial step size $\alpha_{k,l}$ that is acceptable. We approximate a minimum desired step size using linear models of the involved functions

defined as,

$$\alpha_k^{\min} := \gamma_\alpha \cdot \begin{cases} \min\left\{\gamma_\theta, \frac{\gamma_\varphi \theta(x_k)}{-\nabla\varphi_{\mu_j}(x_k)^T d_k}, \frac{\kappa_\theta [\theta(x_k)]^{s_\theta}}{[-\nabla\varphi_{\mu_j}(x_k)^T d_k]^{s_\varphi}}\right\} & \text{if } \nabla\varphi_{\mu_j}(x_k)^T d_k < 0 \\ \gamma_\theta & \text{otherwise} \end{cases} \tag{3.10}$$

with a safety-factor $\gamma_\alpha \in (0, 1]$ and switch the algorithm to the feasibility restoration phase when $\alpha_{k,l}$ becomes smaller than $\alpha_k^{\min}$.

3.1.4 IPOPT algorithm

The overall algorithm for solving the equality constrained NLP (3.1) is stated in Algorithm I.

Algorithm I

Given: Starting point (x_0, λ_0, v_0) with $x_0, v_0 > 0$; initial value for the barrier parameter $\mu_0 > 0$ and $\delta_w^{last} \leftarrow 0$; Constants $\epsilon_{tol}, \kappa_\theta, \kappa_\epsilon > 0$, $s_\theta > 1$, $s_\varphi \geq 1$, $\gamma_\alpha \in (0, 1]$, $\gamma_\varphi, \gamma_\theta \in (0, 1)$, $\eta_\varphi \in (0, \frac{1}{2})$ and $0 < \bar{\delta}_x^{min} < \bar{\delta}_x^{max}$, $\bar{\delta}_c > 0$, $0 < \kappa_x^- < 1 < \kappa_x^+ < \bar{\kappa}_x^+$, $\kappa_c \geq 0$.

1. *Initialize.* Initialize the filter $\mathcal{F}_0 := \{(\theta, f) \in \mathbb{R}^2 : \theta \geq \theta_{\max}\}$ and the iteration counter $k \leftarrow 0$.

2. *Check convergence for the overall problem.* If $E_0(x_k, \lambda_k, v_k) \leq \epsilon_{tol}$ (with the error estimate E_0 defined in (3.4)), then STOP [CONVERGED].

3. *Check convergence for the barrier problem.* If $E_{\mu_j}(x_k, \lambda_k, v_k) \leq \kappa_\epsilon \mu_j$, then

   3.1. Update $\mu_{j+1} = \max\left\{\epsilon_{tol}/10, \min\left\{\kappa_\mu \mu_j, \mu_j^{\theta_\mu}\right\}\right\}$, and set $j \leftarrow j + 1$

   3.2. Re-initialize the filter $\mathcal{F}_0 := \{(\theta, f) \in \mathbb{R}^2 : \theta \geq \theta_{\max}\}$

   3.3. If $k = 0$ repeat step 3, otherwise continue at step 4.

4. *Compute search direction with regularization.*

4.1. Factorize the KKT matrix in (3.8) with $\delta_x = \delta_c = 0$. If the inertia is (n, m, 0), then calculate d_k and go to step 5. Otherwise, continue with step 4.2.

4.2. If the matrix has zero eigenvalues, set $\delta_c = \bar{\delta}_c \mu_j^{\kappa_c}$ Otherwise, set $\delta_c = 0$

4.3. If $\delta_x^{last} = 0$, set $\delta_x = \bar{\delta}_x^0$, otherwise set $\delta_x = \max\left(\bar{\delta_x}^{min}, \kappa_x^- \delta_x^{last}\right)$

4.4. Attempt to factorize the KKT matrix in (3.8) with δ_x and δ_c. If inertia is correct then set $\delta_x^{last} = \delta_x$ and calculate d_k and go to step 5. Otherwise continue with step 4.5

4.5. If $\delta_x^{last} = 0$, set $\delta_x = \bar{\kappa}_x^+ \delta_x$, otherwise set $\delta_x = \kappa_x^+ \delta_x$.

4.6. If $\delta_x > \bar{\delta}_x^{max}$, abort the current step and directly go to step 9. Otherwise, go to step 4.4.

5. *Backtracking line search.*

5.1. *Initialize line search.* Calculate α_k^{max}, set $\alpha_{k,0} = min(\alpha_k^{max}, 1)$ and $l \leftarrow 0$.

5.2. *Compute new trial point.* If the trial step size becomes too small, i.e. $\alpha_{k,l} < \alpha_k^{\min}$ with $\alpha_k^{\min}$ defined by (3.10), go to the feasibility restoration phase in step 9. Otherwise, compute the new trial point $x_k(\alpha_{k,l}) = x_k + \alpha_{k,l} d_k$.

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

5.4. *Check sufficient decrease with respect to current iterate.*

5.4.1. *Case I: $\alpha_{k,l}$ is an f-step-size (i.e. SC holds):* If the AC for the objective function holds, accept the trial step and go to step 6. Otherwise, go to step 5.5.

5.4.2. *Case II: $\alpha_{k,l}$ is not an f-step-size (i.e. SC is not satisfied):* If SDC holds, accept the trial step and go to step 6. Otherwise, go to step 5.5.

5.5. *Choose new trial step size.* Set $\alpha_{k,l+1} = \frac{1}{2}\alpha_{k,l}$, $l \leftarrow l + 1$, and go back to step 5.2.

6. *Accept trial point.* Set $\alpha_k := \alpha_{k,l}$ and $x_{k+1} := x_k(\alpha_k)$.

7. *Augment filter if necessary.* If k is not an f-type iteration, augment the filter using (3.9); otherwise leave the filter unchanged, i.e. set $\mathcal{F}_{k+1} := \mathcal{F}_k$.

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

9. *Feasibility restoration phase.* Compute a new iterate x_{k+1} by decreasing the primal infeasibility, so that x_{k+1} satisfies SDC and is acceptable to the filter. Augment the filter using (3.9) (for x_k) and continue with the regular iteration in step 8.

3.1.5 Restoration phase

Restoration phase (step 9 in Algorithm I) is called in IPOPT to obtain a new iterate $x_{k+1} > 0$ with $x_{k+1} \notin \mathcal{F}_{k+1}$ when the line search fails. Sometimes, it is also used as a final attempt to get a new start pointing when an internal error occurs in the normal phase. The restoration phase algorithm applies the normal phase outlined in the previous sections to a smooth reformulation of the optimization problem

$$\min_{x \in \mathbb{R}^n} \quad \rho \|c(\bar{x})\|_1 + \frac{\zeta}{2} \|D_R(\bar{x} - \bar{x_R})\|_2^2 \quad \text{s.t. } \bar{x} \geq 0. \tag{3.11}$$

Here, we try to minimize the primal feasibility of problems (3.1) and avoid a large deviation from x_R where the restoration phase is called. The weight parameter ζ in the reference point penalty term is set as $\zeta := \sqrt{\mu}$ which is driven to zero by the barrier parameter μ, the scaling matrix D_R is defined by

$$D_R = \text{diag}(\min\{1, 1/|\bar{x}_R^{(1)}|\}, ..., \min\{1, 1/|\bar{x}_R^{(n)}|\}). \tag{3.12}$$

and the scaling parameter $\rho = 1000$ seems to work well in practice.

Introducing non-negative slack variables $\bar{p}, \bar{n} \in \mathbb{R}^m$, we obtain the smooth reformulation

of (3.11) shown as follows,

$$\begin{aligned} \min_{x\in\mathbb{R}^n,\bar{p},\bar{n}\in\mathbb{R}^m} \quad & \rho\sum_{i=1}^{n}(\bar{p}^{(i)}+\bar{n}^{(i)})+\frac{\zeta}{2}\|D_R(\bar{x}-\bar{x}_R)\|_2^2 \\ \text{s.t.} \quad & c(\bar{x})-\bar{p}+\bar{n}=0 \\ & \bar{x},\bar{p},\bar{n}\geq 0 \end{aligned} \tag{3.13}$$

While solving the optimization problem (3.13), we obtain either a point $x_{k+1} \notin \mathcal{F}_{k+1}$ and return to normal phase or fully converge (3.13) to a local minimum. Due to the SC and AC, no feasible point can be present in the filter (see [22] Lemma 4). Therefore, all feasible points should be acceptable to the filter. If the restoration problem (3.13) converges without finding a point acceptable to the filter, then it has not found a feasible point. The point that it converges to is therefore a local minimum of infeasibility close to the reference point x_R. In this case, IPOPT stops in restoration without converging to a KKT point of the original NLP.

3.2 NLP format in IPOPT

Although all NLPs can be reformulated to the standard form (3.1), IPOPT deals with a more complicated form directly for efficiency, which is shown as follows,

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & c(x)=0 \\ & d^L \leq (E_d^L)^T d(x) \quad (E_d^U)^T d(x) \leq d^U \\ & x^L \leq (E_x^L)^T x \qquad (E_x^U)^T x \leq x^U \end{aligned} \tag{3.14}$$

where matrices E_d^L, E_x^L, E_d^U and E_x^U expand the space of the bounds into the full space of the variables or inequalities and d^L, x^L, d^U and x^U are bounds of variables and inequalities. As we mentioned in section 3.1, IPOPT requires all inequalities to be feasible in each

iteration. In that case, an initial point is hard to obtain based on the form (3.14). By introducing the slack varibles s, all inequality constraints become equalities, and it is easy to find an interior point for all inequality constraints. The slacked NLP problem is shown as follows,

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \qquad & d(x) - s = 0 \\ & d^L \leq (E_d^L)^T s \quad & (E_d^U)^T s \leq d^U \\ & x^L \leq (E_x^L)^T x \quad & (E_x^U)^T x \leq x^U \end{aligned} \tag{3.15}$$

3.3 Convergence properties

The global convergence proof of IPOPT shows that every limit point of the sequence of iterates generated by the algorithm is feasible and that there exists at least one limit point that is a stationary point for the problem [22]. The local convergence proof of IPOPT shows a superlinear convergence rate [23]. However, certain assumptions including Linear Independence Constraint Qualification (LICQ), Second-Order Sufficient Conditions (SOSC), and a successful restoration phase are required to prove both properties. However, these assumptions are not always true in real world problems. In particular, IPOPT might have difficulties to solve problems that violate the regularity conditions.

Chapter 4

Structured Regularization for Equality Constraints

One challenging type of problem for IPOPT is solving problems with locally dependent equality constraints. In this chapter, we will propose a structured regularization method to replace the general regularization method we discussed in Chapter 3. Structured elimination can effectively eliminate the dependent equality constraints locally while maintaining the high performance of IPOPT. We applied this method to a modified benchmark NLP library and two blending problems.

4.1 Introduction

Degenerate optimization models (e.g., with dependent equality constraints) arise in many nonlinear programming (NLP) applications. These constraints do not change the optimal solution but may cause NLP solvers to fail because constraint qualifications are violated and the resulting NLP subproblems become ill-conditioned. Degenerate or dependent equality constraints appear in two ways. *Globally dependent* equalities are redundant at all feasible points. These constraints result from naive modeling and may often be deleted directly. *Locally dependent* equalities are defined as constraints that are only redundant at a subset of iterates. In a model with locally dependent constraints, equality constraint gradients are linearly dependent at particular points as a result of non-linearity of the

constraints.

Whereas globally dependent constraints can (and ideally should) be removed when building a model, local degeneracy appears in a number of optimization applications and is harder to avoid. Locally dependent constraints arise often in the following engineering models. First, in water network synthesis and blending problems [24], local degeneracies occur in the continuous relaxation of the mixed-integer nonlinear program (MINLP) [25]. For instance, when a process unit or warehouse is deselected, the total flow rate through the unit vanishes and flow balance constraints are locally degenerate. Moreover, complementarity constraints, which model switching behaviors such as fluid phase disappearance and flow reversal, are dependent at all feasible points and many reformulations of these complementarity constraints still retain locally dependent constraints [26]. Finally, discretized dynamic problems with (high index) path constraints become locally dependent as the finite element mesh is refined [27].

NLP solvers that are based on subproblems that require constraint linearizations often struggle with ill-conditioning that results from dependent constraints. Consequently, these NLP solvers mitigate the effects of dependent constraints in a number of ways. Active set methods [13, 18, 19] formulate and solve subproblems that identify constraints that are predicted to be active at the solution. Dependent constraints may be identified and removed by extending the algorithm's inherent constraint selection mechanism. Sequential quadratic programming (SQP) methods such as SNOPT [13] are observed to have local superlinear convergence without the Linear Independence Constraint Qualification (LICQ) [28], and reduced gradient methods such as CONOPT [18] handle dependent constraints efficiently as a part of the active set selection. However, inefficiency results as the number of constraints and degrees of freedom increase, and the active set choices increase exponentially.

In contrast, interior point methods demonstrate advantages in solving large-scale NLP problems with many degrees of freedom [29], and they remove the need for combinatorial active set selection. However, without a constraint selection mechanism degenerate constraints cannot be removed and convergence difficulties arise, as well as unbounded multiplier estimates [2]. In primal-dual interior point methods [10, 15, 17] dependent equality constraints lead to ill-conditioned KKT systems, which are often treated with general-purpose regularization strategies. However, with regularization the convergence rate deteriorates, and much slower performance and more frequent premature terminations are observed.

This work develops an improved regularization approach for interior point methods for dealing with degenerate models. The remainder of the chapter is structured as follows. Section 2 introduces the primal-dual barrier approach and discusses existing regularization strategies. In Section 3, we propose three structured regularization methods to identify and effectively eliminate degenerate constraints. In Section 4, the IPOPT algorithm is presented with modifications to use structured regularization. The three structured regularization methods are implemented and compared to the current versions of IPOPT [15], KNITRO [10], and CONOPT [18] on a suite of test problems, with results given in Section 5. The last section concludes the chapter and suggests areas for future work.

4.2 IPOPT regularization

As we mentioned in Chapter 3, IPOPT regularization modifies the KKT matrix as follows,

$$\begin{bmatrix} W_k + \Sigma_k + \delta_x I & A_k \\ (A_k)^T & -\delta_c I \end{bmatrix}. \tag{4.1}$$

By introducing the small pivot element δ_c, the degeneracy in the equality constraints is revised in the KKT matrix. Meanwhile δ_x is used to maintain the inertia in W_k. In every iteration, the LBL^T factorization of matrix (4.1) with $\delta_x = \delta_c = 0$ is attempted. If it succeeds, we use the resulting search direction from Equation (3.8) as the Newton step. Otherwise, δ_c is set to $\bar{\delta}_c \mu_j^{\kappa_c}$ in (4.1) where $\bar{\delta}_c$ is a parameter defaulted at 10^{-8} and κ_c is a positive floating point value defaulted at 0.25.

After δ_c is fixed, δ_x increases step by step until the inertia is correct or δ_x exceeds its upper bound $\bar{\delta}_x^{max}$ (defaulted at 10^{20}). δ_c decreases and is forced to zero with μ_j during the solving process. It has been proved that for any $\delta_c > 0$, there exist suitable values of δ_x such that the matrix (4.1) has the correct inertia (n, m, 0) [7].

Although the inertia can be corrected, setting the parameter $\bar{\delta}_c$ must be done carefully in practice. If $\bar{\delta}_c$ is chosen too small for pivoting, the inertia is not corrected; hence δ_x is increased repeatedly and may become very large, leading to failure in the Newton step. As a result, the solver switches to feasibility restoration phase immediately. On the other hand, a larger $\bar{\delta}_c$ may have a large influence on all the equality constraints and distorts the Newton step. As a result, the best choice for $\bar{\delta}_c$ is problem dependent.

The IPOPT regularization is based on the assumption that equality constraints are linearly independent at the optimal point, which is not always the case. One possible scenario is the case of globally dependent constraints, i.e. constraints that are redundant for all feasible points. Two strategies are embedded in IPOPT to deal with global dependence. One is an option which turns on the global dependency detector as a part of the presolve phase. The detector generates several random initial points within variable bounds and calculates the Jacobian matrix at these points. If a constraint is dependent at all points, it is treated as a globally dependent constraint and removed before the solving phase. The other heuristic is a structured degeneracy switch, which is active if δ_c is needed in several

consecutive iterations. In that situation, δ_c is always added before the factorization of KKT matrix without testing the degeneracy of the original KKT matrix. This method can save a factorization step in each iteration and reduce the effect of failure in detecting singular KKT matrices. But meanwhile, because of the small regularization term, the Newton step does not lie perfectly in the null space of Jacobian matrix, which delays the convergence rate. Finally, both methods aim at reducing the influence of global dependence and may not help local dependence. Instead, we propose three efficient structured regularization methods to eliminate dependent constraints, both locally and globally.

4.3 Structured Regularization

To solve the linear system (3.8), an LBL^T factorization is applied to the KKT matrix. During the IPOPT regularization, only inertia information is retained and all partial factorization results are discarded. However, from the indices of the unpivoted columns in the LBL^T factorization, the indices of the dependent constraints can be determined. In structured regularization, we use this information in alternative modifications to the KKT matrix.

The multifrontal method, a direct method for LBL^T factorization of sparse linear systems, was proposed by Duff and Reid [30] in 1983 and attracted great interest due to its low memory utilization and high efficiency. Popular sparse linear system solvers including HSL_MA57 [31], HSL_MA97 [32], MUMPS [33] and WSMP [34] are all based on this method. Other linear solvers (e.g, PARDISO [35]) that use a direct method for LBL^T factorization can also retrieve the required structural information.

There are three phases in multifrontal method linear solvers. First, in the *analysis phase*, the solver accepts the sparsity pattern of the KKT matrix in compressed sparse column

format or coordinate format. Then the solver determines the elimination sequence, analyzes the sparsity pattern of the matrix, and prepares the data structures for the next step. Note that the analysis phase is generally applied only once for each NLP problem since the structure of KKT matrix remains unchanged throughout the optimization, and because the preprocessing time is significant.

Second, the *factorization phase* exploits the data structures set up by the analysis phase to compute a sparse LBL^T factorization of the KKT matrix where L is a lower triangular matrix and B is a block diagonal matrix with blocks of order 1 or 2 on its diagonal. A user specialized threshold $\xi \in \mathbb{R}^+$ is used to maintain the stability of the factorization. For each pivot a_{l_1,l_1}, we check the 1×1 stability test which is defined as follows,

$$|a_{l_1,l_1}| > \xi \times \max_{\bar{l} \geq l_1+1} |a_{l_1,\bar{l}}|. \tag{4.2}$$

If test (4.2) holds, a_{l_1,l_1} is used as a pivot. Otherwise, the 2×2 stability test

$$\left| \begin{pmatrix} a_{l_1,l_1} & a_{l_1,l_2} \\ a_{l_2,l_1} & a_{l_2,l_2} \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{\bar{l} \geq l_1+2} |a_{l_1,\bar{l}}| \\ \max_{\bar{l} \geq l_1+2} |a_{l_2,\bar{l}}| \end{pmatrix} \leq \begin{pmatrix} \xi^{-1} \\ \xi^{-1} \end{pmatrix} \tag{4.3}$$

is applied, where $|\cdot|$ indicates the (component-wise) absolute value. If the test (4.3) also fails, the pivot a_{l_1,l_1} is delayed by permuting the matrix to attempt a different pivot. It has been proved that for any pivot threshold $\xi \in (0, 0.5]$, a pivot sequence of 1×1 or 2×2 pivots can always be obtained if the matrix is non-singular [36]. If all the remaining pivot candidates are delayed pivots that fail the stability test, the solver reports that the matrix is singular, with the unfinished factorization results in its working array. The structure of the unfinished working array is exploited in Section 4.3.3. The factorization phase is the most time consuming step per IPOPT iteration. It is applied only once in each iteration if there is no regularization. When regularization is applied, a new factorization is required whenever the terms (δ_x, δ_c) change. This dramatically slows down the optimization pro-

cess.

Third, the *solution phase* is the simplest phase in the linear solver. It calculates the solution using backsolves, according to the factorization results and the user provided right hand sides. In IPOPT, the factorization may be reused several times per iteration, since second order correction and refinement steps will only change the right hand side.

In IPOPT, the analysis phase is only called once per NLP solution since the sparsity pattern of the KKT matrix is assumed to remain the same. In structured regularization, we focus on the case where the factorization results can attribute the incorrect inertia to the presence of degenerate constraints. Ideally, the degenerate constraints should be removed from the KKT matrix, but this would require a new analysis phase for the linear solver. Instead, we compare modifications to the existing KKT matrix structure that can effectively (numerically) eliminate degenerate constraints.

Moreover, based on the factorization information from the linear solver, we can determine if an incorrect inertia is due to the Jacobian or the Hessian. This allows us to fix the degeneracy in the Jacobian first. Then, if the incorrect inertia is not attributed to degenerate constraints (e.g. due to an indefinite reduced Hessian), the δ_x term in (3.8) is used for regularization.

4.3.1 Big-M regularization

A preliminary version of a structured regularization method was proposed in [37], which introduces big-M terms instead of the regularization with δ_c. Assuming A_k has column rank $m - r$, the Jacobian matrix A_k can be separated as $[(A_k)_I|(A_k)_D]$ with dependent (or even nearly dependent) columns in $(A_k)_D$ based on the index of dependent rows, so $[c_I(x_k)|c_D(x_k)]$ and $[(d_k^\lambda)_I|(d_k^\lambda)_D]$ may also be separated accordingly in the linear system.

Here we modify KKT matrix in (3.7) as follows,

$$\begin{bmatrix} W_k + \Sigma_k & (A_k)_I & (A_k)_D \\ (A_k)_I^T & 0 & 0 \\ (A_k)_D^T & 0 & -M \end{bmatrix} \begin{pmatrix} d_k^x \\ (d_k^\lambda)_I \\ (d_k^\lambda)_D \end{pmatrix} = - \begin{pmatrix} \nabla\varphi_\mu(x_k) + A_k\lambda_k \\ c_I(x_k) \\ c_D(x_k) \end{pmatrix} \tag{4.4}$$

where $M = \delta_D I$ is an $r \times r$ diagonal matrix with δ_D a large constant.

The linearized equations are divided in two parts,

$$\begin{aligned} (A_k)_I^T d_k^x &= -c_I(x_k) \\ (A_k)_D^T d_k^x - \delta_D (d_k^\lambda)_D &= -c_D(x_k) \end{aligned} \tag{4.5}$$

Note there is no change in the equations corresponding to independent constraints, but a new entity δ_D is added to the dependent constraints. By assigning $\delta_D >> (A_k)_D^T\ d_k^x + c_D(x_k)$, $(d_k^\lambda)_D$ approaches zero, which eliminates the effect of dependent constraints in (4.4). Thus, the linear system of the Newton step (3.7) is nearly equivalent to

$$\begin{bmatrix} W_k + \Sigma_k & (A_k)_I \\ (A_k)_I^T & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ (d_k^\lambda)_I \end{pmatrix} = - \begin{pmatrix} \nabla\varphi_\mu(x_k) + A_k\lambda_k \\ c_I(x_k) \end{pmatrix}. \tag{4.6}$$

This concept was tested on a FORTRAN IPOPT implementation in [37] using the MUMPS linear solver.

4.3.2 Constraint elimination regularization

Similar to the idea of big-M regularization, a constraint elimination regularization is proposed by replacing $(A_k)_D^T$ and $c_D(x_k)$ by zeroes and adding the negative identity matrix at the diagonal corresponding to dependent constraints. Then the modified KKT matrix is shown as follows,

$$\begin{bmatrix} W_k + \Sigma_k & (A_k)_I & 0 \\ (A_k)_I^T & 0 & 0 \\ 0 & 0 & -I \end{bmatrix} \begin{pmatrix} d_k^x \\ (d_k^\lambda)_I \\ (d_k^\lambda)_D \end{pmatrix} = - \begin{pmatrix} \nabla\varphi_\mu(x_k) + A_k\lambda_k \\ c_I(x_k) \\ 0 \end{pmatrix}. \tag{4.7}$$

In the limit as $M \to +\infty$, the linear system solution of big-M regularization (4.4) is equivalent to the solution of (4.7). Moreover, constraint elimination regularization avoids numerical difficulties associated with introducing very large terms in the factorization, but it also loses the flexibility of adjusting δ_D to include linear constraints that are nearly dependent.

4.3.3 On the fly regularization

In the two previously presented structured regularization methods, we regularize the KKT matrix and re-factorize it again. However, after the modification, the pivot sequence may be changed in the new matrix, and it is possible that there are new null pivots reported in the second factorization. Also as we mentioned before, the factorization is a very expensive step. In that case, we propose the "on the fly" regularization. This method changes the working arrays of the unfinished factorization directly, which makes them identical to the factorization results of the constraint elimination regularization, and therefore allows these working arrays to be used directly in the solving phase.

Assume the LBL^T factorization of the KKT matrix with dependent constraints is shown as follows,

$$P \begin{bmatrix} L & 0 \\ X & I \end{bmatrix} \begin{bmatrix} B_I & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L & 0 \\ X & I \end{bmatrix}^T P^T \tag{4.8}$$

where P is the permutation matrix and L is the lower triangular matrix with the order $n + m - r$. X corresponds to the dependent constraints that are the unfinished part of the factorization, as a result of the delayed null pivot steps. To regularize the KKT matrix, we simply modify the results of LBL^T factorization (4.8) to the form,

$$P \begin{bmatrix} L & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} B_I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & I \end{bmatrix}^T P^T. \tag{4.9}$$

In (4.9), we delete X and add a negative identity matrix with order r to the diagonal matrix.

This modified factorization results in the same factorization as for the KKT matrix in constraint elimination regularization, i.e. Equation (4.7). In the linear solver, the factorization results are compressed in a working array, detailed in Section 4.4.3. This working array is modified on the fly by regularization, and this leads to structural changes corresponding to (4.9). While the previous two approaches require a separate factorization for structured regularization, on-the-fly regularization factorizes and regularizes the KKT matrix in a single step, which leads to fewer overall factorizations. Nevertheless, this method depends on the selection of dependent constraints from the LBL^T factorization. If the pivot threshold is too small for the matrix or the matrix is badly scaled, very small pivots may remain in the B_I matrix. In the previous two regularization methods, a dependent threshold ϵ_D is set by users, and all the pivots in B are scanned and compared to ϵ_D, with all pivots less than ϵ_D taken as the null pivots. However, as the factorization is applied only once in the "on the fly" elimination, sequential pivoting and application of stability test may still introduce very small pivots in the B_I matrix, which may lead to unstable steps in the solving process.

4.4 Modified IPOPT Regularization

In this section, we introduce the full IPOPT algorithm with the proposed modifications.

4.4.1 IPOPT algorithm with Structured Regularization

Algorithm II

Given: Starting point (x_0, λ_0, v_0) with $x_0, v_0 > 0$; initial value for the barrier parameter $\mu_0 > 0$ and $\delta_x^{last} \leftarrow 0$; Constants $\epsilon_{tol}, \kappa_\epsilon > 0$, $s_{max} \geq 1$, $s_\theta > 1$, $\kappa_\theta > 0$, $\kappa_\mu \in (0,1)$, $\theta_\mu \in (1,2)$, $s_\varphi \geq 1$, $\gamma_\varphi, \gamma_\theta \in (0,1)$, $\eta_\varphi \in (0, \frac{1}{2})$ and $0 < \bar{\delta}_x^{min} < \bar{\delta}_x^{max}$, $0 < \kappa_x^- < 1 < \kappa_x^+ < \bar{\kappa}_x^+$,

$\theta_{max} > 0$, $\alpha^{min} \in (0, 1)$.

1. *Initialize.* Initialize the filter $\mathcal{F}_0 := \{(\theta, f) \in \mathbb{R}^2 : \theta \geq \theta_{\max}\}$ and the iteration counter $k \leftarrow 0$.

2. *Check convergence for the overall problem.* If $E_0(x_k, \lambda_k, v_k) \leq \epsilon_{tol}$ (with the error estimate E_0 defined in (3.4)), then STOP [CONVERGED].

3. *Check convergence for the barrier problem.* If $E_{\mu_j}(x_k, \lambda_k, v_k) \leq \kappa_\epsilon \mu_j$, then

   3.1. Update $\mu_{j+1} = \max\left\{\epsilon_{tol}/10, \min\left\{\kappa_\mu \mu_j, \mu_j^{\theta_\mu}\right\}\right\}$, and set $j \leftarrow j + 1$

   3.2. Re-initialize the filter $\mathcal{F}_0 := \{(\theta, f) \in \mathbb{R}^2 : \theta \geq \theta_{\max}\}$

   3.3. If $k = 0$ repeat Step 3, otherwise continue at Step 4.

4. *Compute search direction with regularization.*

   4.1. Factorize the KKT matrix in (3.8) with $\delta_x = 0$. If the inertia is (n, m, 0), then calculate d_k and go to Step 5. If the matrix has at least one zero eigenvalue, then continue with Step 4.2. Otherwise, go to Step 4.5

   4.2. Choose the structured regularization method:

      4.2.1. For big-M and constraint elimination regularizations: Apply the regularization term following either equation (4.4) or (4.7). Continue with Step 4.3.

      4.2.2. For on-the-fly regularization: modify the results of LBL^T factorization results and inertia of the matrix, go to Step 4.4.

   4.3. Factorize the KKT matrix with structured regularization term.

   4.4. If the inertia is (n, m, 0), then calculate d_k and go to Step 5. Otherwise continue with Step 4.5.

   4.5. If $\delta_x^{last} = 0$, set $\delta_x = \bar{\delta}_x^0$, otherwise set $\delta_x = \max\left(\bar{\delta_x}^{min}, \kappa_x^- \delta_x^{last}\right)$

4.6. Attempt to factorize the KKT matrix with δ_x and the regularization modification. If inertia is correct then set $\delta_x^{last} = \delta_x$, calculate d_k and go to Step 5. Otherwise continue with Step 4.7

4.7. If $\delta_x^{last} = 0$, set $\delta_x = \bar{\kappa}_x^+ \delta_x$, otherwise set $\delta_x = \kappa_x^+ \delta_x$.

4.8. If $\delta_x > \bar{\delta}_x^{max}$, abort the current step and directly go to Step 9 of Algorithm II. Otherwise, go to Step 4.6.

5. *Backtracking line search.*

5.1. *Initialize line search.* Calculate α_k^{max}, set $\alpha_{k,0} = min(\alpha_k^{max}, 1)$ and $l \leftarrow 0$.

5.2. *Compute new trial point.* If the trial step size becomes too small, i.e. $\alpha_{k,l} < \alpha^{min}$, go to the feasibility restoration phase in Step 9. Otherwise, compute the new trial point $x_k(\alpha_{k,l}) = x_k + \alpha_{k,l} d_k$.

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

5.4. *Check sufficient decrease with respect to current iterate.*

5.4.1. *Case I: $\alpha_{k,l}$ is an f-type step-size (i.e. SC holds):* If AC holds, go to Step 6. Otherwise, go to Step 5.5.

5.4.2. *Case II: $\alpha_{k,l}$ is not an f-type step-size (i.e. SC is not satisfied):* If SDC holds, go to Step 6. Otherwise, go to Step 5.5.

5.5. *Choose new trial step size.* Set $\alpha_{k,l+1} = \frac{1}{2}\alpha_{k,l}$, $l \leftarrow l + 1$, and go back to Step 5.2.

6. *Accept trial point.* Set $\alpha_k := \alpha_{k,l}$ and $x_{k+1} := x_k(\alpha_k)$.

7. *Augment filter if necessary.* If k is not an f-type iteration, augment the filter using (3.9); otherwise leave the filter unchanged, i.e. set $\mathcal{F}_{k+1} := \mathcal{F}_k$.

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

9. *Feasibility restoration phase.* Compute a new iterate x_{k+1}, so that x_{k+1} satisfies SDC and is acceptable to the filter. Augment the filter using (3.9) (for x_k) and continue with the regular iteration in Step 8.

While all the structured regularization methods eliminate the degenerate columns of A_k, adjustment of δ_x in Step 4 of Algorithm II is still necessary to maintain the correct inertia and guarantee a descent direction. Also, in Step 4.4 of Algorithm II, there should be no zero eigenvalues attributed to the Jacobian part of the KKT matrix. However, it is numerically possible that nearly dependent constraints are reported after structured regularization, even if they were not detected before. For this reason, in the implementation a small value of δ_c can still be applied in Step 4.4 even if degenerate constraints are detected.

4.4.2 Convergence properties of IPOPT

The global convergence proof of IPOPT shows that every limit point of the sequence of iterates generated by the algorithm is feasible and that there exists at least one limit point that is a stationary point for the problem [22]. The local convergence proof of IPOPT shows a superlinear convergence rate [23] under Second-Order Sufficiency Conditions (SOSC). Certain assumptions including LICQ and the successful restoration phase are required to prove both properties. However, these assumptions do not always hold on real world problems. In particular, IPOPT might have difficulties solving problems that violate the regularity conditions.

The global convergence proofs for IPOPT in [22] may be extended to prove convergence when LICQ holds *after* structured regularization. Note that in this section the Lemmas reference the global convergence paper [22].

For column rank of A_k equal to $m-r$, we determine the $m-r$ independent columns in A_k and form a new matrix A_k^R. Also, a column selection matrix is denoted as $\eta_k \in \mathbb{R}^{m\times(m-r)}$ to map independent columns in A_k to the full column rank matrix A_k^R. Then c_k^R can be defined correspondingly:

$$A_k^R = A_k\eta_k \tag{4.10a}$$

$$c_k^R = \eta_k^T c(x_k). \tag{4.10b}$$

Based on our structured regularization method, Equation (3) in [22] can be written analogously to (3.7) as follows,

$$\begin{bmatrix} H_k & A_k^R \\ A_k^{R^T} & 0 \end{bmatrix} \begin{pmatrix} d_k \\ \lambda_k^{R+} \end{pmatrix} = - \begin{pmatrix} g_k \\ c_k^R \end{pmatrix}. \tag{4.11}$$

where $H_k = W_k + \Sigma_k$ and $g_k = \nabla\varphi_{\mu_j}(x_k)$.

Although linear system (3.7) may have multiple solutions if A_k is degenerate, we know that the KKT conditions are still consistent. In that case, λ_k^{R+} can be extended by adding 0 to the corresponding unselected r columns and the extended vector should be one of the solutions for λ_k in Equation (3) in [22], i.e. $\lambda_k^{R+} := \eta_k^T\lambda_k$. Note also that if A_k has full column rank, then $A_k^R = A_k$ and $r = 0$, so (4.11) above is equivalent to (3) in [22].

Now we list the assumptions for proving global convergence adapted from [22] with slight modifications.

Assumptions G. *Let $\{x_k\}$ be the sequence generated by Algorithm II, where we assume that the feasibility restoration phase in Step 9 always terminates successfully and that the algorithm does not stop in Step 2 at a KKT point. We denote $\mathcal{R}_{\text{inc}}$ as the set of iteration counters, in which the restoration phase is invoked from Step 4.6 of Algorithm II.*

(G1) *There exists an open set $\mathcal{C} \subseteq \mathbb{R}^n$ with $[x_k, x_k + d_k] \subseteq \mathcal{C}$ for all $k \notin \mathcal{R}_{\text{inc}}$, so that f and c are differentiable on $\mathcal{C}$, and their function values, as well as their first derivatives, are bounded and Lipschitz-continuous over $\mathcal{C}$.*

(G2) *The matrices H_k approximating the Hessian of the Lagrangian in (3) (from [22]) and η_k are uniformly bounded for all $k \notin \mathcal{R}_{\text{inc}}$.*

(G3) *The Hessian approximations H_k are uniformly positive definite on the null space of the Jacobian A_k^T. In other words, there exists a constant $M_H > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$*

$$eig_{\min}\left(Z_k^T H_k Z_k\right) \geq M_H, \tag{4.12}$$

where the columns of $Z_k \in \mathbb{R}^{n\times(n-m+r)}$ form an orthonormal basis matrix of the null space of A_k^T. Similarly, we denote $Y_k \in \mathbb{R}^{n\times(m-r)}$ as an orthonormal basis matrix for the range space of A_k^T.

(G4) *There exists a constant $M_A > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$ we have*

$$\sigma_{\min}(A_k^R) \geq M_A, \tag{4.13}$$

where $\sigma_{\min}(A_k^R)$ represents the smallest singular value of A_k^R.

(G5) *The iterates, for which the restoration phase is invoked from Step 4 (for example, when (4.12) or (4.13) are violated), are not arbitrarily close to the feasible region. In other words, there exists a constant $\theta_{\text{inc}} > 0$, so that $k \notin \mathcal{R}_{\text{inc}}$ whenever $\theta(x_k) \leq \theta_{\text{inc}}$.*

(G6) *The linear system (3.7) has at least one solution for all $k \notin \mathcal{R}_{\text{inc}}$.*

The decomposition of the overall search direction can be defined as,

$$\bar{q}_k := -\left[A_k^{R^T} Y_k\right]^{-1} c_k^R \tag{4.14a}$$

$$\bar{p}_k := -\left[Z_k^T H_k Z_k\right]^{-1} Z_k^T \left(g_k + H_k Y_k \bar{q}_k\right) \tag{4.14b}$$

$$d_k = Z_k \bar{p}_k + Y_k \bar{q}_k \tag{4.14c}$$

and we define two criticality measures for convergence, $\chi(x_k) = \|\bar{p}_k\|$ and $\theta(x_k) = \|c(x_k)\|$.

The first two lemmas of the global convergence proof are presented below, with modifications to accommodate structured regularization.

Lemma 1. *Suppose Assumptions G hold. Then there exist constants M_d, M_λ, $M_m > 0$, such that*

$$\|d_k\| \leq M_d, \qquad \|\lambda_k^{R+}\| = \|\lambda_k^{+}\| \leq M_\lambda, \qquad |m_k(\alpha)| \leq M_m \alpha \tag{4.15}$$

for all $k \notin \mathcal{R}_{\text{inc}}$ and $\alpha \in (0, 1]$.

Proof. From (G1) we have that the right hand side of (4.11) is uniformly bounded. Additionally, Assumptions (G2), (G3), and (G4) guarantee that the inverse of the matrix in (4.11) exists and is uniformly bounded for all $k \notin \mathcal{R}_{\text{inc}}$. Consequently, the solution of (4.11), (d_k, λ_k^{R+}), is uniformly bounded, and therefore also $m_k(\alpha)/\alpha = g_k^T d_k$. Since λ_k^+ is an extension of λ_k^{R+}, the solution of (3) in [22] is also uniformly bounded. □

The following result shows that the search direction is a direction of sufficient descent for the objective function at points that are sufficiently close to feasibility but still not sufficiently optimal.

Lemma 2. *Suppose Assumptions G hold. If $\{x_{k_i}\}$ is a subsequence of iterates for which $\chi(x_{k_i}) \geq \epsilon$ with a constant $\epsilon > 0$ independent of i then there exist constants $\epsilon_1, \epsilon_2 > 0$, such that*

$$\theta(x_{k_i}) \leq \epsilon_1 \qquad \Longrightarrow \qquad m_{k_i}(\alpha) \leq -\epsilon_2 \alpha.$$

for all i and $\alpha \in (0, 1]$.

Proof. Consider a subset $\{x_{k_i}\}$ of iterates with $\chi(x_{k_i}) = \|\bar{p}_{k_i}\|_2 \geq \epsilon$. Then, by Assumption (G5), for all x_{k_i} with $\theta(x_{k_i}) \leq \theta_{\text{inc}}$ we have $k_i \notin \mathcal{R}_{\text{inc}}$. Furthermore, with $\bar{q}_{k_i} = O(\|c^R(x_{k_i})\|)$ (from (4.14a) and Assumption (G4)) and $\|c^R(x_{k_i})\| = O(\|c(x_{k_i})\|)$ (from (4.10b)

and Assumption (G2)) it follows that for $k_i \notin \mathcal{R}_{\text{inc}}$

$$\begin{aligned} m_{k_i}(\alpha)/\alpha &= g_{k_i}^T d_{k_i} \overset{(4.14c)}{=} g_{k_i}^T Z_{k_i} \bar{p}_{k_i} + g_{k_i}^T Y_{k_i} \bar{q}_{k_i} && (4.16a) \\ &\overset{(4.14b)}{=} -\bar{p}_{k_i}^T \left[Z_{k_i}^T H_{k_i} Z_{k_i} \right] \bar{p}_{k_i} - \bar{p}_{k_i}^T Z_{k_i}^T H_{k_i} Y_{k_i} \bar{q}_{k_i} + g_{k_i}^T Y_{k_i} \bar{q}_{k_i} && (4.16b) \\ &\overset{(G2),(G3)}{\leq} -c_1 \left\| \bar{p}_{k_i} \right\|_2^2 + c_2 \left\| \bar{p}_{k_i} \right\|_2 \left\| c^R(x_{k_i}) \right\| + c_3 \| c^R(x_{k_i}) \| && (4.16c) \\ &\leq \chi(x_{k_i}) \left(-\epsilon \; c_1 + c_2 \theta(x_{k_i}) + \frac{c_3}{\epsilon} \theta(x_{k_i}) \right) && (4.16d) \end{aligned}$$

for some constants $c_1, c_2, c_3 > 0$, where we used $\chi(x_{k_i}) \geq \epsilon$ in the last inequality. If we now define

$$\epsilon_1 := \min \left\{ \theta_{\text{inc}}, \frac{\epsilon^2 \; c_1}{2(c_2 \; \epsilon + c_3)} \right\},$$

it follows for all x_{k_i} with $\theta(x_{k_i}) \leq \epsilon_1$ that

$$m_{k_i}(\alpha) \leq -\alpha \frac{\epsilon \; c_1}{2} \chi(x_{k_i}) \leq -\alpha \frac{\epsilon^2 \; c_1}{2}.$$

The claim follows after defining $\epsilon_2 := \frac{\epsilon^2 \; c_1}{2}$. □ □

Lemmas 3 - 10 from [22] hold without modification, so the convergence theorem follows,

Theorem 4. *Suppose Assumptions G hold. Then*

$$\begin{aligned} \lim_{k \to \infty} \theta(x_k) &= 0 && (4.17a) \\ \textit{and} \qquad \liminf_{k \to \infty} \chi(x_k) &= 0. && (4.17b) \end{aligned}$$

In other words, all limit points are feasible, and if $\{x_k\}$ is bounded, then there exists a limit point x_* of $\{x_k\}$, which is a first order optimal point for the equality constrained NLP (1) in [22].

4.4.3 Implementation

The main part of the structured regularization implementation is to determine the indices of the dependent constraints from the linear solvers and (for on-the-fly regularization) modify the results of factorization directly as described in Section 4.3.3.

MUMPS [33] provides a very user friendly interface for the indices of dependent columns/rows. By setting a dependent threshold *thres*, MUMPS returns a list of "null pivots" which are smaller than *thres* by a vector *PIVNTL_LIST*. The default $thres$ is set as $thres = \epsilon \times 10^{-5} \times \|A_{pre}\|_\infty$, where ϵ is the machine precision (2.22×10^{-16} for the test computer) and $\|A_{pre}\|_\infty$ is the permuted and scaled matrix to be factorized. In addition, $thres$ can be set to either an absolute value or a relative value proportional to $\|A_{pre}\|_\infty$. We used the absolute threshold with our methods.

MA97 [32] provides a function *ma97_enquire_indef* to obtain the information on the pivot sequence and the matrix B^{-1} in a LBL^T factorization. Since the stability test [36] prevents small pivots in 2×2 blocks, we can set $thres$ and compare it with the pivots in all 1×1 blocks. If a pivot is smaller than $thres$, we look for the pivot sequence and add the index to the null pivot indices list.

MA57 [31] has no interface for the factorization information. After calling the factorization phase, all the information is organized by frontal matrices and compressed in the working area array. We wrote a routine to uncompress the frontal matrices one by one and extract the B matrices in (4.8). Then we can analyse the matrix in the same way as we did with MA97. Another routine is implemented for on-the-fly regularization to change the working array directly and send it back to the solve phase.

4.5 Numerical results

We implemented the structured regularization methods as an option in IPOPT 3.12.6 and ran the test cases on an Intel(R) Xeon(R) CPU E5-2440 @ 2.40GHz×12 with 64 GiB memory. KNITRO 10.1.1 and CONOPT 3.17A are used as the comparison solvers. IPOPT is compiled with GNU Fortran 4.8.4 and GCC 4.8.4 with the suggested BLAS library. HSL

for IPOPT and MUMPS 4.10.0 are used as linear solvers. For structured regularization, $\delta_D = 10^{30}$ in (4.4) and $thres = 10^{-10}$ are default options. Also, the tolerance options tol is set as 10^{-5} and the rest of the options are all at default value. All the test problems are formulated in AMPL and compiled without the presolve step.

For comparing solvers, we set *rtredg* in CONOPT and *opttol* in KNITRO to 10^{-5}, corresponding to the choice of tol in IPOPT. To provide stable timings, we solved all problems multiple times so that the total CPU time exceeds 2 CPU seconds for each problem; then an average CPU time was computed and used as the solution time for each problem.

4.5.1 Toy example

We motivate the structured regularization for IPOPT with familiar mass balance equations [38]. Many chemical process models can be scaled up by increasing the number of units or adding reaction equations as constraints inside the units. Defining streams $s \in S$, components $c \in C$ and units $u \in U$, leads to the following mass balance constraints

$$\begin{aligned} F_s &= \sum_{c \in C} f_{s,c} \qquad && s \in S \\ f_{s,c} &= x_{s,c} F_s \qquad && c \in C, s \in S \\ \sum_{s \in In_u} f_{s,c} &= \sum_{s \in Out_u} f_{s,c} && c \in C, \end{aligned} \tag{4.18}$$

where F_s, $f_{s,c}$, and $x_{s,c}$ are total flow rate, component flow rate, and mole fraction, respectively. In_u and Out_u are the input and output stream sets for unit u. We deliberately add the fraction summation equations,

$$\sum_{c \in C} x_{s,c} = 1, \tag{4.19}$$

which are dependent with constraints (4.18).

Figure 4.1 shows a test example with two process units in series, five process streams, and three components in each stream. Each unit has one inlet and two outlets. One of the

output flows in the first unit is the second unit's input flow. We maximize the component flow rate of s_2, subject to (4.18) and (4.19).

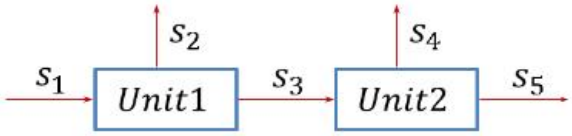

Figure 4.1: Flowsheet for toy example

Using the IPOPT regularization with the MUMPS linear solver [33], IPOPT fails on the second iteration due to the failure in factorizing the KKT matrix. This situation is mentioned in Section 4.2, where δ_c is set too small (10^{-8}) and δ_x increases until it reaches its upper bound, 10^{20}. The solver then switches to the feasibility restoration phase. Since the initial point is a feasible point and the purpose of restoration is to reduce infeasibility, calling the restoration phase at a feasible point terminates the optimization.

However with the big-M structured regularization, the five dependent constraints are correctly detected and eliminated in each iteration. Therefore, the Newton steps are well-defined and the algorithm converges to an optimal solution. Note that in structured regularization, IPOPT with MUMPS and MA57 share the same convergence path, which is identical to the convergence path without the globally dependent constraints (4.19). The results are shown in Table 4.1, along with a comparison to CONOPT and KNITRO.

4.5.2 Nonlinear blending problems

Blending problems are an example of the type of model exhibiting local degeneracy. The blending problem is common in refinery processes and always occurs when the product requirement cannot be met by a single source. Thus it is very important to know the best way to mix different feeds from the refinery with maximum profit, subject to the quality

or property requirements of the different final products.

The general gasoline blending formulation is defined for products $p \in P$, feed sources $i \in I$, and intermediate tanks $q \in Q$ over a time horizon with N_t time periods, $t \in \{1, \ldots, N_t\}$, as follows [7],

$$
\begin{aligned}
\max \quad & \sum_{t \in T} \left(\sum_{p \in P} cost_p s_{t,p} - \sum_{i \in I} cost_i s_{t,i} \right) \\
s.t. \quad & \sum_{p \in P} s_{t,qp} - \sum_{i \in I} s_{t,iq} + \nu_{t+1,q} = \nu_{t,q}, \ t \in \{1, \ldots, N_t\}, q \in Q \\
& \sum_{i \in I} \omega_{t,i} s_{t,iq} - \sum_{p \in P} \omega_{t,q} s_{t,qp} + \omega_{t,q} \nu_{t,q} = \omega_{t+1,q} \nu_{t+1,q}, \ t \in \{1, \ldots, N_t - 1\}, q \in Q \\
& \omega_{t,p} s_{t,p} - \sum_{q \in Q} \omega_{t,qp} s_{t,qp} = 0, \ t \in 1, \ldots, N_t, p \in P \\
& \omega_{t,i} s_{t,i} - \sum_{q \in Q} \omega_{t,iq} s_{t,iq} = 0, \ t \in 1, \ldots, N_t, i \in I,
\end{aligned}
\tag{4.20}
$$

where the indexed variables $s_{t,lm}$ represent a stream flow between tank indices l and m, and $\omega_{t,l}$ and $\nu_{t,l}$ are qualities (i.e. blend stream properties) and inventories for index l, respectively at time t.

Although the nonlinear blending models have proprietary data, the key characteristic lies in the bilinear, nonconvex terms. When particular flows in the blending network are set to zero during the solution steps, the blending equations become degenerate and dependent linearized constraints are formed.

Two industrial examples from [37] are considered. CONOPT and KNITRO can solve *blend1* easily, but for *blend2*, CONOPT fails to find a feasible point and KNITRO makes poor progress for many iterations before terminating with an optimal solution. Meanwhile, for the current IPOPT, both blending problems exceed the maximum iterations because the regularization terms distort Newton steps too much and slow down the rate of convergence. If this maximum iteration limit is increased, these problems may even-

tually be solved. On the other hand, all of the structured methods detect and eliminate the dependent constraints efficiently and easily solve both cases. On the larger problem *blend2*, IPOPT with structured regularization is the fastest solver. Further characteristics and numerical results are presented in Table 4.1.

4.5.3 CUTEr test set

The CUTEr test set [39] is a well known collection of NLP problems from various academic sources, used as a benchmark NLP library. To test our structured regularization methods, we select 227 problems, which have both equality constraints and enough degrees of freedom to add a new redundant constraint. The selected model set is called the original problem set. Then for all the selected models, a new nonlinear constraint $c_1(x) - c_1^2(x) = 0$ is added as the last constraint, which is globally dependent with the first constraint $c_1(x) = 0$. Experience reported with this additional constraint is also reported in [40].

The results are represented by performance plots proposed by Dolan and More [41]. Define $t_{pr,m}$ as a performance characteristic (iteration count here) of a structured regularization method m in solving the problem pr. Then the performance scaled by the best method is defined as follows,

$$r_{pr,m} = \frac{t_{pr,m}}{\min_{m'} t_{pr,m'}} \tag{4.21}$$

with the performance profile for each method defined as,

$$\pi_m(\tau) = \frac{\#\ of\ problems\ with\ r_{pr,m} \leq \tau}{total\ \#\ of\ problems}, \quad \tau \geq 1. \tag{4.22}$$

With MA57, iteration counts are compared for the current version of IPOPT for original and modified problem sets, and also for the structured regularization methods for modified problem results are shown in Figure 4.2.

From Figure 4.2, we can see that the current version of IPOPT works very well with the

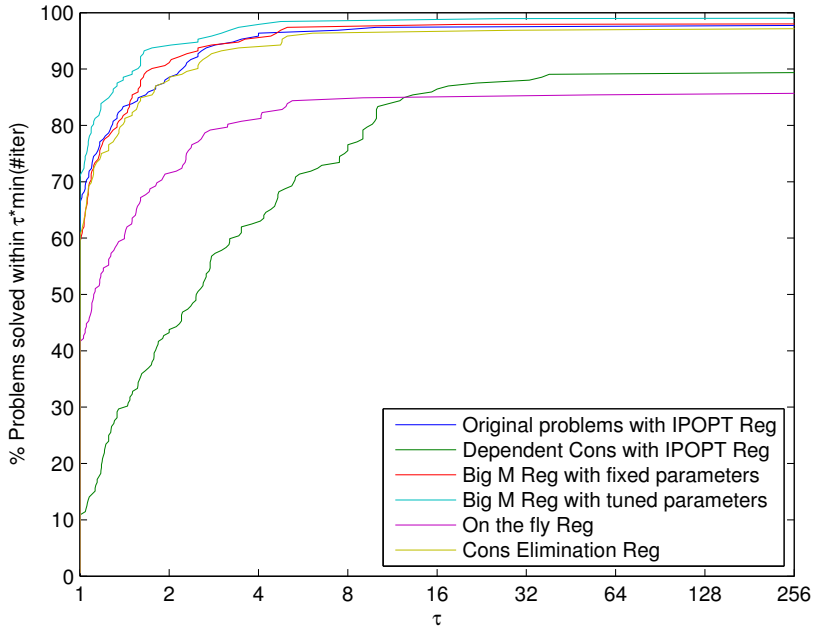

Figure 4.2: Iteration Count Performance of MA57

original problem set. However, if the new dependent constraint is added, there is a huge drop in performance, which illustrates the limitations of current IPOPT regularization in solving problems with dependent constraints.

Here, the "on the fly" regularization method performs well for the small problems. But as the size of problem increases, more errors in detecting dependent rows are made by the linear solver, which dramatically affects the performance of the "on the fly" regularization. Nevertheless, because the on-the-fly method performs the factorization and structured regularization at the same time, it requires the fewest regularizations for the problems it can solve.

The performance profiles of constraint elimination regularization (4.7) and big M regularization (4.4) with fixed parameters are very close. As we mentioned before, a problem

dependent threshold is introduced to both of the methods, which is used to correct the unstable factorization results. The default of this parameter (*thres*) is 10^{-6}. If this parameter is tuned well for a specific problem in big-M regularization, the performance is much better. Interestingly, it is even better than the current version of IPOPT for the original problems. This is because there is still local dependence in several problems such as *steenbrc*, *steenbrd*, *steenbre*, and *steenbrf*, even though all problems in the original problem set are well posed. The current version of IPOPT fails on all of them, while the structured regularization methods can handle this local dependence.

Similar performance profiles are shown in the Figures 4.3 and 4.4 with linear solver MA97 and MUMPS. Only the results of big M regularization are presented here since it is the most robust among the three structured regularization methods. We observe from Figures 4.2, 4.3, and 4.4 that the structured method is relatively sensitive to the dependent threshold parameter, which should be tuned carefully for different problems.

The performances of current IPOPT, IPOPT with big-M regularization (tuning parameters), KNITRO and CONOPT have been compared on the CUTEr test set. Performance profiles are shown in Figure 4.5 and Figure 4.6. The performance metric is CPU time in Figure 4.5 and iteration count in Figure 4.6.

CONOPT reports 47 problems as locally infeasible, which are counted as failures in our tests. However, it is very fast on the smaller problems in the test set. KNITRO is slightly faster than the current version of IPOPT, although the proportions of problems finally solved are similar. However, IPOPT with structured regularization is the fastest and most stable of the considered solvers.

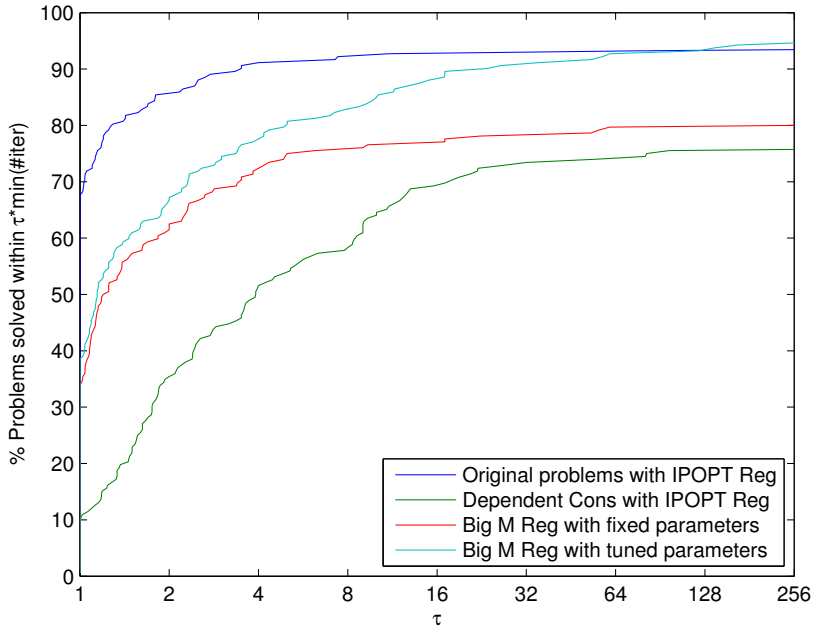

Figure 4.3: Iteration Count Performance of MA97

4.6 Conclusion

In this chapter we have proposed three structured regularization methods for IPOPT to deal with dependent equality constraints. In general, big-M regularization is the most robust and is recommended first for problems for which degeneracy is a suspected difficulty. The performance may be improved by tuning the $thres$ parameter to adjust to the specific problem scaling. In contrast, on-the-fly regularization is not as robust but requires the fewest factorizations for small problems with dependencies.

Future work will focus on dependent inequality constraints, since structured regularization can only help with dependent equality constraints. With dependent active inequality constraints, the barrier NLP solvers may still have trouble solving the problem. NLP reformulations of Mathematical Programs with Complementarity Constraints (MPCCs) are

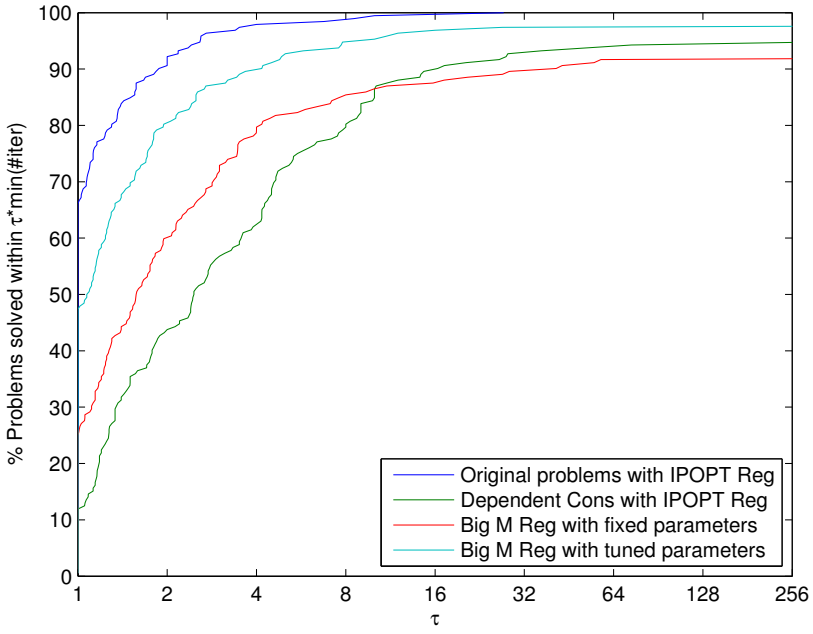

Figure 4.4: Iteration Count Performance of MUMPS

a common example of this challenge. For MPCC, MFCQ is violated at any feasible point and inequalities are involved in the degenerate active set. This problem will be addressed further in Chapter 6

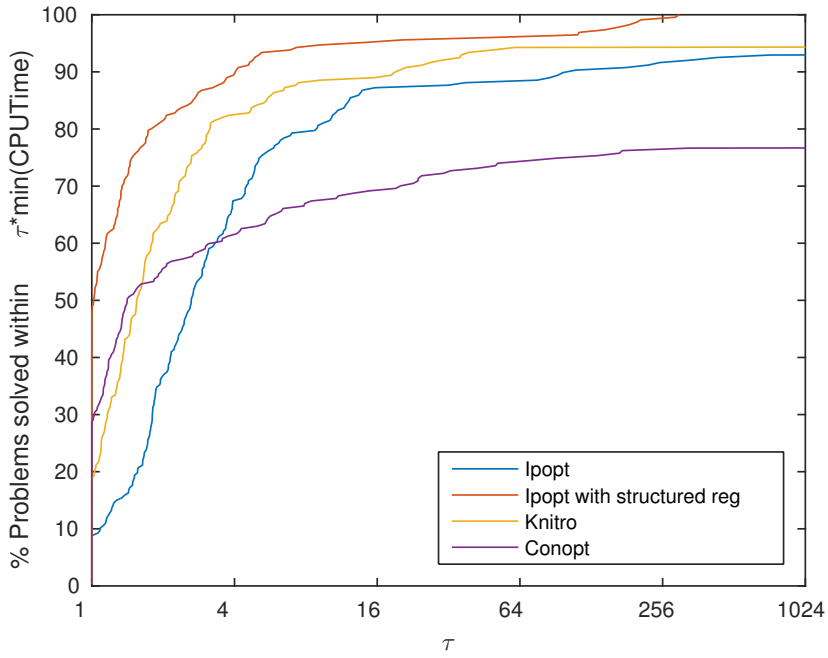

Figure 4.5: CPU Time Comparison between IPOPT, KNITRO and CONOPT

Table 4.1: Result comparison on toy example and blending problems

	Toy example	**Blend1**	**Blend2**
#var	32	827	5142
#con	31	772	4702
CONOPT	Optimal solution found #iter=26 CPU time=0.009s	Optimal solution found #iter=47 CPU time=0.061s	Locally infeasible #iter=1939
KNITRO	Optimal solution found #iter=4 CPU time=0.013s	Optimal solution found #iter=18 CPU time=0.162s	Optimal solution found #iter=1101 CPU time=259.864s
IPOPT-MUMPS	Restoration failed #iter=2	Maximum iterations #iter$\geq$3000	Maximum iterations #iter$\geq$3000
IPOPT-MA57	Optimal solution found #iter=14 CPU time=0.020s	Maximum iterations #iter$\geq$3000	Maximum iterations #iter$\geq$3000
IPOPT-MUMPS Big-M	Optimal solution found #iter=12 CPU time=0.018s	Optimal solution found #iter=48 CPU time=0.754s	Optimal solution found #iter=714 CPU time=171.650s
IPOPT-MA57 Big-M	Optimal solution found #iter=12 CPU time=0.012s	Optimal solution found #iter=31 CPU time=0.304s	Optimal solution found #iter=501 CPU time=118.230s

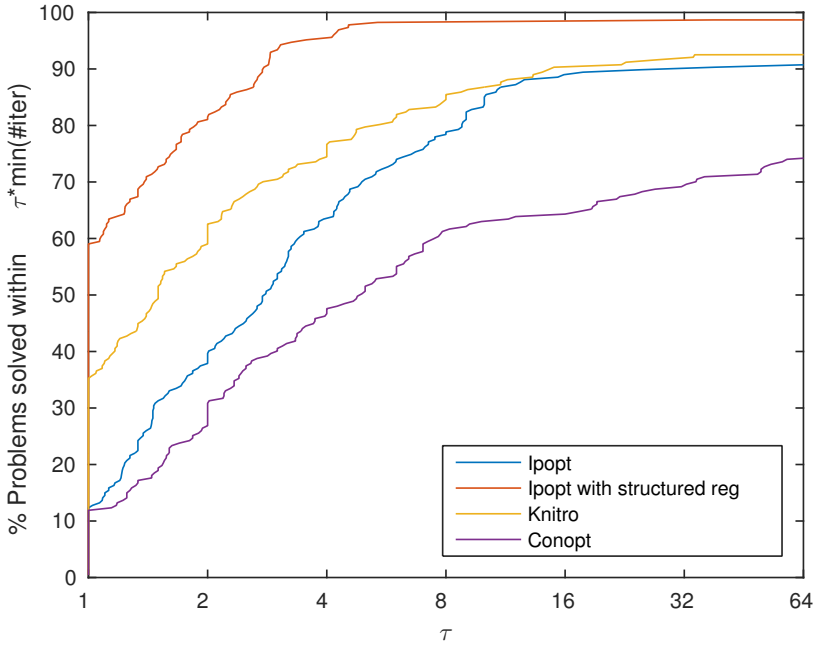

Figure 4.6: Iteration Comparison between IPOPT, KNITRO and CONOPT

Chapter 5

Parallel Cyclic Reduction Decomposition for Dynamic Optimization Problems

In this chapter, we focus on improving the speed of IPOPT for solving dynamic optimization problems. We take advantage of the special problem structure in dynamic optimization and apply cyclic reduction to calculate the KKT linear system in parallel. Several variants of the cyclic reduction are compared, and the best method shows significant parallel speed up. The method is further applied to four applications to demonstrate the convergence of the NLPs.

5.1 Introduction

Optimization of dynamic systems has been shown to be very useful in many areas of science and engineering. Common examples include state estimation, parameter estimation, and optimal control. These problems are often applied in real-time, in a rolling horizon framework [42]. Therefore, we need fast and robust numerical algorithms for solving these problems.

Methods to solve dynamic optimization problems can be divided into two groups: indirect or variational methods derived from Pontryagin's Maximum principle [43, 44], and direct methods that make use of nonlinear programming (NLP) solvers. Indirect methods use the calculus of variations to write the necessary optimality conditions as a boundary

value problem. By directly using optimality conditions, indirect methods can produce highly accurate solutions, but the boundary problems can be difficult to solve and efficiently handling state constraints remains a challenge.

Direct methods may be further divided into sequential and simultaneous methods. Sequential methods discretize only the control variables and then use an NLP solver to optimize the parameterized control variables [45, 46]. Sensitivity information from the dynamic system is obtained by integrating the direct sensitivity equations or the adjoint equations [47]. In sequential methods, the entire time horizon is integrated with a fixed set of control parameters along with senstivity profiles. In general, sequential methods have the advantage of using reliable solvers for differential algebraic systems (DAEs). However, the repeated solution of these systems can become very time consuming and they may fail for unstable dynamic systems.

Multiple shooting is a hybrid of simultaneous and sequential methods, in which the time domain is divided into several smaller segments and constraints are added to the NLP to enforce continuity between the time segments. Sensitivity information is then calculated with respect to both the control parameters and the initial conditions within each time segment.

Simultaneous methods discretize the entire problem and embed the discretized states and controls in a large-scale nonlinear program. The typical discretization uses collocation, which is a fully implicit Runge-Kutta method. These methods have high order accuracy and good stability properties. However, improving accuracy corresponds to adding more discretization points, which can quickly grow to form very large NLPs. Nevertheless, the NLPs remain very sparse and have additional structure that we will explore in this work.

One method to improve the computational efficiency of dynamic optimization is to reduce the model size through model order reduction [48, 49]. By reducing the dimension-

ality of the dynamic system before optimization, the computational effort should be reduced. However, accuracy of the reduced models remains a problem. In addition, the reduced models may lose advantageous features such as sparsity that make the original system more amenable to optimization [49].

Several strategies to reduce the computational cost of dynamic optimization look to parallelization. In the category of sequential methods, researchers look for ways to parallelize the sensitivity calculation from the dynamic system [50]. For multiple shooting methods, the integration and sensitivity calculation in each time interval is easily parallelizable [51, 52, 53].

By contrast, efforts to use parallelization in the simultaneous approach must focus on parallelizing the nonlinear programming solver. In existing nonlinear programming software, sparse linear solvers are able to take advantage of slight parallelism in the basic floating point operations when factoring or backsolving. However, the overall nonlinear programming algorithm remains serial. To achieve significant benefit from parallelism, we need to parallelize the linear solver algorithm. Parallel linear algebra methods normally focus on specific matrix structures. It has been observed that the KKT matrix in simultaneous dynamic optimization can be permuted to a block tridiagonal structure. This has been exploited by several researchers using Schur complement decomposition [54, 55, 56, 57, 58]. In our previous work [59], we compared the Schur complement decomposition and cyclic reduction, and found that cyclic reduction was a more promising method for general dynamic optimization problems. In this work, we will further investigate cyclic reduction for dynamic optimization and demonstrate the performance on several case studies.

Cyclic reduction (CR) was proposed by Hockney and Golub in 1965 [60] in the context of numerical solutions of partial differential equations. The review paper by Gander and Golub [61] covers several developments and applications of the method. Although CR is

observed to be fast, issues with numerical stability were also observed. An algorithmic variant with stability guarantees was proposed by Yalamov and Pavlov [62]. In this chapter, we examine this method in detail and compare to traditional cyclic reduction on the KKT matrix structure found in simultaneous methods for dynamic optimization.

In this work, we first review the basic form of dynamic optimization problems and the simultaneous method for solving them in section 5.2. Then, a description of traditional CR and Yalamov's variant is given in section 5.3 with implementation details in section 5.4. In sections 5.5 and 5.6, these methods are applied to generic tridiagonal matrices and tridiagonal systems of the structure found in dynamic optimization. In section 5.7, the best parallel method is applied within an NLP solver for four case studies and the results are compared to the same solver using serial linear algebra routines. Conclusions are given in section 5.8 with discussion of directions for future work.

5.2 Dynamic Optimization

We start with a general dynamic optimization problem stated in the following form:

$$\begin{aligned} \min \quad & \int_{t_0}^{t_F} \hat{\Phi}(z,y,u)dt + \hat{\Psi}(z(t_F)) \\ s.t. \quad & \frac{dz}{dt} = g(z,y,u) \\ & h(z,y,u) = 0 \\ & z(t_0) = \bar{z_0} \\ & z^L \leq z \leq z^U \\ & y^L \leq y \leq y^U \\ & u^L \leq u \leq u^U \end{aligned} \tag{5.1}$$

where unknowns z, y, u are all functions of time $t \in [t_0, t_F]$. z is a vector of differential state variables, y is a vector of algebraic variables, and u is a vector of control, or input,

variables. $\hat{\Phi}$ is the integrand for the objective function, such as set point tracking in control problems. $\hat{\Psi}$ is the terminal cost at the end of the process. The semi-explicit differential-algebraic equation model is assumed to be index 1. g gives the right hand side of the differential equations, and h are algebraic equation constraints. All these constraints are required to hold for all time $t \in [t_0, t_F]$.

To solve problem (5.1) with a simultaneous approach (also known as direct transcription) we fully discretize the state and control variables. This can lead to large-scale NLP problems. We will focus attention of collocation methods, where the time horizon is discretized to N finite elements with K collocation points in each finite element. Define the index set of collocation points within a finite element $\mathcal{C} = \{1 \dots K\}$ and the finite element set $\mathcal{F} = \{1 \dots N\}$. The discretized version of problem (5.1) is given by:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{N-1} \hat{\Phi}(z_i, y_i, u_i) + \hat{\Psi}(z_N) \\
s.t. \quad & z_1 = \bar{z_0} \\
& z_{i+1} = z_{i,K}, \;\; i = 1, \dots, N-1 \\
& \dot{z}_{i,j} = g(z_{i,j}, y_{i,j}, u_{i,j}), \\
& z_{i,j} = z_i + \sum_{k=1}^{K} w_{k,j} \dot{z}_{i,k} \\
& h(z_{i,j}, y_{i,j}, u_{i,j}) = 0 \\
& z^L \leq z_i, z_{i,j} \leq z^U \\
& y^L \leq y_{i,j} \leq y^U \\
& u^L \leq u_{i,j} \leq u^U \\
& i = 1, \dots, N \quad j = 1, \dots, K
\end{aligned} \tag{5.2}$$

we define variables $\dot{z}_{i,j}$ representing the derivative of the states, and $z_{i,j}$ representing the value of the states. These are indexed over $i \in \mathcal{F}$ and $j \in \mathcal{C}$. The collocation equations use parameters w, which are derived from the orthogonal polynomial representation of

the states [7]. These parameters are related to both the choice of orthogonal polynomials and the length of the finite elements.

The size of problem (5.2) (numbers of variables and constraints) is directly proportional to $N \times K$. This means that higher accuracy requires larger optimization problems. This leads to a significant challenge for NLP solvers to solve these problems in a reasonable computation time.

In this section we first present some background material on interior point methods for solving NLPs. These methods are well suited for large-scale NLP problems that we face in dynamic optimization. We then discuss how to take advantage of the special structure of discretized dynamic optimization problems (5.2) to improve the performance of interior point methods.

5.2.1 Interior Point Methods

In Chapter 3, we reviewed the IPOPT algorithm. To solve the linear system (3.7), IPOPT applies a symmetric linear solver, such as HSL_MA57 [31], HSL_MA97 [32], MUMPS [33], PARDISO [35] or WSMP [34]. It first applies a LBL^T factorization to the KKT matrix on the left side of (3.7), followed by backsolves to obtain the solution. These solvers are well-known to be stable and fast. However, solving this linear system (3.7) is the most time consuming step in IPOPT. To speed up the performance of IPOPT on very large problems, we can look to speed up the linear solver through parallelization.

We refer to linear system (3.7) as the KKT system, and the matrix on the left hand side as the 'KKT Matrix'. The solution vector of the linear system is simply referred to as variables, and the right hand side of this linear system is referred to as 'rhs'. For the rest of this chapter, we focus only on the linear solve within a single iteration. Moreover we assume the KKT matrix is well-posed, and the inertia is $(n, m, 0)$. (Recall that inertia is a triple

consisting of the number of positive eigenvalues, the number of negative eigenvalues, and the number of zero eigenvalues.) Regularization of the KKT matrix can be applied to make this condition hold, and several procedures are described in [7, 15].

5.2.2 Exploiting Structure in KKT System

For the discretized dynamic optimization problem (5.2), we recall that i is the index of finite elements and we define $w_i^T = [\dot{z}_{i,j}^T, z_{i,j}^T, y_{i,j}^T, u_{i,j}^T]$. Now (5.2) can be simplified as follows,

$$\begin{aligned} \min \quad & \sum_{i=1}^{N-1} \hat{\Phi}(z_i, w_i) + \hat{\Psi}(z_N) \\ s.t. \quad & G(z_{i+1}, w_i) = G_i(w_i) - z_{i+1} = 0 \;\; i = 1, \ldots, N-1 \\ & H_i(z_i, w_i) = 0, \;\; i = 1, \ldots, N \end{aligned} \tag{5.3}$$

Comparing (5.3) to (5.2), we group all constraints related only to one finite element i to $H_i(z_i, w_i)$ including the ODEs, algebraic equations, and collocation equations. Then we group all continuity equations between adjacent finite elements. Note that the variable bounds have been added as barrier terms and do not affect the structural analysis in this section. In addition, the initial condition for z_1 is substituted directly into the problem, so the first constraint of (5.2) is eliminated implicitly.

If IPOPT is applied to solve the simplified discretized dynamic optimization problem (5.3), the KKT linear system (3.7) is shown as follows,

$$\begin{bmatrix} W_{zz} & W_{zw} & \nabla_z H & \nabla_z G \\ W_{wz} & W_{ww} & \nabla_w H & \nabla_w G \\ \nabla_z H^T & \nabla_w H^T & 0 & 0 \\ \nabla_z G^T & \nabla_w G^T & 0 & 0 \end{bmatrix} \begin{pmatrix} \Delta z \\ \Delta w \\ \Delta \lambda^H \\ \Delta \lambda^G \end{pmatrix} = - \begin{pmatrix} r_z \\ r_w \\ r_H \\ r_G \end{pmatrix} \tag{5.4}$$

where $H^T = [H_1^T, H_2^T, \ldots, H_N^T]$ and $G^T = [G_1^T - z_2^T, G_2^T - z_3^T, \ldots, G_{N-1}^T - z_N^T]$. The rhs (r_z, r_w, r_H, r_G) corresponding to different variables can be derived from (3.7). $\Delta\lambda^H$ and

$\Delta\lambda^G$ are steps for the multipliers corresponding to constraints H and G. These can be partitioned by the corresponding finite elements as λ_i^H and λ_i^G, giving the overall variable ordering

$$[\Delta z_2^T, \ldots, \Delta z_N^T, \Delta w_1^T, \ldots, \Delta w_N^T, \Delta\lambda_1^{H^T}, \ldots, \Delta\lambda_N^{H^T}, \Delta\lambda_1^{G^T}, \ldots, \Delta\lambda_N^{G^T}] \tag{5.5}$$

However, if we reorder the variable vector by grouping together all primal and dual variables corresponding to the same finite elements, we get

$$[\Delta w_1^T, \Delta\lambda_1^{H^T}, \Delta\lambda_1^{G^T}, \Delta z_2^T, \Delta w_2^T, \Delta\lambda_2^{H^T}, \Delta\lambda_2^{G^T}, \ldots, \Delta z_N^T, \Delta w_N^T, \Delta\lambda_N^{H^T}, \Delta\lambda_N^{G^T}] \tag{5.6}$$

The KKT matrix corresponding with this rearranged variable vector has the following structure,

$$\begin{bmatrix}
\boxed{\begin{array}{ccc|cccc}
W_{w_1w_1} & \nabla_{w_1}H_1 & \nabla_{w_1}G_1 & & & & \\
\nabla_{w_1}H_1^T & 0 & 0 & & & & \\
\nabla_{w_1}G_1^T & 0 & 0 & -I & & & \\
\hline
 & & -I & W_{z_2z_2} & W_{z_2w_2} & \nabla_{z_2}H_2 & \nabla_{z_2}G_2 \\
 & & & W_{w_2z_2} & W_{w_2w_2} & \nabla_{w_2}H_2 & \nabla_{w_2}G_2 \\
 & & & \nabla_{z_2}H_2^T & \nabla_{w_2}H_2^T & 0 & 0 \\
 & & & \nabla_{z_2}G_2 & \nabla_{w_2}G_2^T & 0 & 0
\end{array}} & & & \\
 & \ddots & -I & & & & \\
 & -I & \boxed{\begin{array}{ccc}
W_{z_Nz_N} & W_{z_Nw_N} & \nabla_{z_N}H_N \\
W_{w_Nz_N} & W_{w_Nw_N} & \nabla_{w_N}H_N \\
\nabla_{z_N}H_N^T & \nabla_{w_N}H_N^T & 0
\end{array}} &
\end{bmatrix} \tag{5.7}$$

where I is the identity matrix. We outline the block pattern for emphasis. Matrix (5.7) is almost block diagonal except for the off-diagonal identity matrices. These come from the continuity constraints that couple the dynamic profile from one time finite element to the next.

Because z_1 is fixed and z_{N+1} is absent, the first and last diagonal blocks are smaller than the rest. The first diagonal block has more constraints than variables and is rank deficient; other block diagonal matrices might be singular as well. Treatment of these cases will be discussed in section 5.3.5.

5.3 Block Cyclic Reduction

Cyclic Reduction has been shown to be a powerful algorithm for solving block tridiagonal linear systems. The basic idea of the algorithm is to eliminate half the unknowns, regroup the equations, and again eliminate half of the unknowns. The process is applied recursively and can be easily parallelized by a large variety of architectures [63]. The general form of a block tridiagonal linear system is shown as follows,

$$\begin{bmatrix} D_1 & U_1 & & & & & \\ L_2 & D_2 & U_2 & & & & \\ & L_3 & D_3 & U_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & L_{N-1} & D_{N-1} & U_{N-1} & \\ & & & & L_N & D_N & \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_{N-1} \\ r_N \end{pmatrix} \tag{5.8}$$

If we compare (5.8) to (5.7), the matrices D_i are the KKT matrices for the individual finite elements, while U_i and L_i only have one identity block on lower left/upper right corner. Note that D_i is always square and symmetric.

When introducing the algorithm, we will assume D_i are nonsingular and defer further discussion about singularity of D_i to section 5.3.5. We also assume that the number of blocks (number of finite elements) N is a power of 2 and not less than 4. This allows us to greatly simplify the derivation and notation. However, the basic algorithm as well as the

time complexity analysis are easily extended to handle arbitrary N.

5.3.1 Traditional Cyclic Reduction

Traditional Cyclic Reduction(TCR) reorders both variables and rhs of the linear system (5.8) in order of odd indices and even indices. The whole matrix is permuted as follows,

$$\left[\begin{array}{cccc|cccc} D_1 & & & & U_1 & & & \\ & D_3 & & & L_3 & U_3 & & \\ & & \ddots & & & \ddots & \ddots & \\ & & & D_{N-1} & & & L_{N-1} & U_{N-1} \\ \hline L_2 & U_2 & & & D_2 & & & \\ & \ddots & \ddots & & & \ddots & & \\ & & L_{N-2} & U_{N-2} & & & D_{N-2} & \\ & & & L_N & & & & D_N \end{array}\right] \left(\begin{array}{c} x_1 \\ x_3 \\ \vdots \\ x_{N-1} \\ \hline x_2 \\ \vdots \\ x_{N-2} \\ x_N \end{array}\right) = \left(\begin{array}{c} r_1 \\ r_3 \\ \vdots \\ r_{N-1} \\ \hline r_2 \\ \vdots \\ r_{N-2} \\ r_N \end{array}\right) \tag{5.9}$$

Now the matrix is separated to four parts where the upper-left and lower-right sub-matrices are both block diagonal. Since D_i are assumed to be nonsingular, we can pivot on the odd diagonals (the upper left section of the matrix in (5.9)) to eliminate the lower right section of the matrix. After doing this, the linear system becomes (5.10) with fill-in blocks marked with X:

$$\left[\begin{array}{cccc|cccc} D_1 & & & & U_1 & & & \\ & D_3 & & & L_3 & U_3 & & \\ & & \ddots & & & \ddots & \ddots & \\ & & & D_{N-1} & & & L_{N-1} & U_{N-1} \\ \hline 0 & 0 & & & X & X & & \\ & \ddots & \ddots & & X & \ddots & \ddots & \\ & & \ddots & 0 & & \ddots & \ddots & X \\ & & & 0 & & & X & X \end{array}\right] \left(\begin{array}{c} x_1 \\ x_3 \\ \vdots \\ x_{N-1} \\ \hline x_2 \\ \vdots \\ x_{N-2} \\ x_N \end{array}\right) = \left(\begin{array}{c} r_1 \\ r_3 \\ \vdots \\ r_{N-1} \\ \hline X \\ \vdots \\ X \\ X \end{array}\right) \tag{5.10}$$

Note here that the fill in pattern in the lower right section is also block tridiagonal. We can apply the same permutation and elimination process recursively until the block size equals one.

Once the block size equals one, we solve the resulting small system. This result gives half of unknowns $[x_2, x_4, \ldots, x_N]$ for the previous iteration, so we simply multiply them by the upper right part of the matrix (5.9) and backsolve for the unknowns with odd indices. This process steps upward until the final solution vector is recovered.

For the implementation, we only need to store the individual matrix blocks and calculate the variable eliminations in parallel. For each cycle k within the TCR recursion, we first calculate three auxiliary matrices for the odd rows, listed as follows:

$$\begin{aligned} P_{2i-1}^{(k)} &= (D_{2i-1}^{(k-1)})^{-1} L_{2i-1}^{(k-1)}; \\ Q_{2i-1}^{(k)} &= (D_{2i-1}^{(k-1)})^{-1} U_{2i-1}^{(k-1)}; \\ Y_{2i-1}^{(k)} &= (D_{2i-1}^{(k-1)})^{-1} r_{2i-1}^{(k-1)}; \end{aligned} \tag{5.11}$$

where i is the block index. To calculate these three matrices, only one LDL^T factorization is applied to each odd diagonal D_{2i-1}. The factors are used for three backsolves, then the

remaining matrices are updated based on P, Q and Y:

$$\begin{aligned} L_i^{(k)} &= -L_{2i}^{(k-1)} P_{2i-1}^{(k)}; \\ U_i^{(k)} &= -U_{2i}^{(k-1)} Q_{2i+1}^{(k)}; \\ D_i^{(k)} &= D_{2i}^{(k-1)} - L_{2i}^{(k-1)} Q_{2i-1}^{(k)} - U_{2i}^{(k-1)} P_{2i+1}^{(k)}; \\ r_i^{(k)} &= r_{2i}^{(k-1)} - L_{2i}^{(k-1)} Y_{2i-1}^{(k)} - U_{2i}^{(k-1)} Y_{2i+1}^{(k)}; \end{aligned} \tag{5.12}$$

Note that some parts of the update rely on auxiliary matrices $2i-1$ and some are related to $2i+1$. Therefore a synchronization point is required after calculating the auxiliary matrices, which will slightly impede parallel performance. Otherwise, the updates for each i may be calculated in parallel, with synchronization only for incrementing cycle count k and after calculating the auxiliary matrices.

At each cycle k, the matrices $P_{2i-1}^{(k)}$, $Q_{2i-1}^{(k)}$, and $Y_{2i-1}^{(k)}$ are retained in memory. After $\log_2(N)$ cycles, the final block size is equal to one and that linear system is solved. Then, the values of $x_i^{(k)}$ are calculated recursively from $k = \log_2(N)$ to $k = 0$ using the stored P, Q, and Y matrices as follows:

$$\begin{aligned} x_{2i-1}^{(k-1)} &= x_i^{(k)}; \\ x_{2i}^{(k-1)} &= Y_{2i}^{(k)} - P_{2i-1}^{(k)} x_i^{(k)} - Q_{2i+1}^{(k)} x_{i+1}^{(k)}; \end{aligned} \tag{5.13}$$

The values $x_i^{(0)}$ will then be the solutions of (5.9). Also, for each update step in the recursion on k, the i's may be updated in parallel. Nevertheless, synchronization is needed when moving from one cycle k to the next.

5.3.2 Yalamov Cyclic Reduction

Yalamov's Cyclic Reduction (YCR) method [62] requires that all L_i, U_i, D_i be square matrices. By constructing a sequence of matrices $V^{(k)}$ and multiplying them to both sides of linear system (5.8), the off-block matrices are pushed further and further away from the

diagonal matrices. When there are only diagonal matrices left, linear solves can be applied block by block in parallel.

The matrices $V^{(k)}$ are constructed as follows. At cycle k, we first generate $P_i^{(k)}$ and $Q_i^{(k)}$,

$$\begin{aligned} P_i^{(k)} &= -L_i^{(k-1)}[D_{i-2^{k-1}}^{(k-1)}]^{-1} \\ Q_i^{(k)} &= -U_i^{(k-1)}[D_{i+2^{k-1}}^{(k-1)}]^{-1} \end{aligned} \tag{5.14}$$

and then use them to construct the matrix $V^{(k)}$

$$V^{(k)} = \begin{bmatrix} I & 0 & \cdots & 0 & Q_1^{(k)} & & & & \\ 0 & \ddots & & & & \ddots & & & \\ \vdots & & \ddots & & & & \ddots & & \\ 0 & & & \ddots & & & & \ddots & \\ P_{2^{k-1}+1}^{(k)} & & & & \ddots & & & & Q_{N-2^{k-1}}^{(k)} \\ & \ddots & & & & \ddots & & & 0 \\ & & \ddots & & & & \ddots & & \vdots \\ & & & \ddots & & & & \ddots & 0 \\ & & & & P_N^{(k)} & 0 & \cdots & 0 & I \end{bmatrix} \tag{5.15}$$

When we apply this matrix to both sides of the linear system (5.8), we get a new linear system with a block diagonal and off-diagonal block bands 2^k blocks away from the diagonal. This process is repeated until the system is fully block diagonal.

Instead of storing the whole KKT matrix, we only manipulate the non-zero blocks. The update formulas are listed as follows, where the matrices $U_i^{(k)}, D_i^{(k)}, L_i^{(k)}, r_i^{(k)}$ represent the upper off-diagonal band matrices, the diagonal matrices, the lower off-diagonal band

matrices, and the right hand sides, respectively:

$$\begin{aligned} L_i^{(k)} &= P_i^{(k)} L_{i-2^{k-1}}^{(k-1)} \\ U_i^{(k)} &= Q_i^{(k)} U_{i+2^{k-1}}^{(k-1)} \\ D_i^{(k)} &= D_i^{(k-1)} + P_i^{(k)} U_{i-2^{k-1}}^{(k-1)} + Q_i^{(k)} L_{i+2^{k-1}}^{(k-1)} \\ r_i^{(k)} &= r_i^{(k-1)} + P_i^{(k)} r_{i-2^{k-1}}^{(k-1)} + Q_i^{(k)} r_{i+2^{k-1}}^{(k-1)} \end{aligned} \tag{5.16}$$

With TCR a synchronization is required after calculating the auxiliary matrices. With YCR the updates of $U_i^{(k)}, D_i^{(k)}, L_i^{(k)}, r_i^{(k)}$ only depend on the auxiliary matrices that match in index i. This means that within one cycle k there is no need for synchronization between processors. Consequently, at each cycle the calculations for each block index i can be done in parallel.

In summary, the algorithm works as follows: at cycle $k = 1 \ldots \log_2(N)$, apply (5.14) and calculate $P_i^{(k)}$ and $Q_i^{(k)}$. Then using (5.16) update $U_i^{(k)}, D_i^{(k)}, L_i^{(k)}, r_i^{(k)}$ and increase the cycle count k by 1. At cycle $\log_2(N)$, only the diagonal matrices are left. So, in the final cycle $k = \log_2(N) + 1$, we compute

$$x_i = [D_i^{(k-1)}]^{-1} r_i^{(k-1)}, i = 1, \ldots, N \tag{5.17}$$

to obtain the solution of (5.8).

5.3.3 Time complexity analysis

Time complexity of an algorithm quantifies the amount of time required by an algorithm in the worst case as a function of the size of the problem. Here we analyze the time complexity of these two cyclic reduction methods based on the number of blocks N. In this sense, we treat the size of blocks as a constant, and quantify scaling in terms of the number of finite elements.

As mentioned before, the LDL^T factorizations are the most time consuming steps in the cyclic reduction algorithm. For the complexity analysis, we assume these steps dominate

the CPU time and simply count the number of factorizations required for a linear system with N blocks. This will be used as a measure of the time complexity. In section 5.5.2, we show that experimental results match the time complexity analysis, which confirms our assumption that number of factorizations dominates the computational expense.

Both algorithms require $\log_2(N)$ cycles, but we need also consider the number of factorizations per cycle. In Yalamov cyclic reduction, $N - 2^{k-1}$ factorizations are performed at the k^{th} cycle, and N factorizations are performed in the last cycle to calculate the solution. So for a tridiagonal linear system with N blocks on the diagonal, $N(\log_2(N) - 1) + 1$ factorizations are performed. Then we can conclude that the time complexity is $O(N \log N)$. In traditional cyclic reduction, only $N/2^k$ factorizations are applied at the k^{th} cycle. So for a tridiagonal linear system with N blocks on the diagonal and $\log_2(N)$ cycles, N factorizations are performed in total. So the time complexity is $O(N)$. If N is not a power of 2, the total number of cycles is $\lceil \log_2(N) \rceil$ and the big O results are still valid.

Based on the complexity analysis, we expect the Yalamov cyclic reduction method to be slower than TCR. However, Yalamov cyclic reduction is more parallelizable than TCR in that it requires fewer synchronizations. If the number of processors is very large (exceeds block number N), then the bottleneck of the algorithm is the number of cyclic reduction iterations instead of the number of factorizations. As shown above, the iteration requirement is the same for the two methods, so we expect performance would be more competitive in a highly parallel environment. However in this study we will only work with 8 processors. The comparison of numerical performance is presented and discussed in section 5.5.

5.3.4 Symmetry

In sections 5.3.1 and 5.3.2, we discussed cyclic reduction methods for general tridiagonal matrices. On the other hand, KKT matrices are symmetric and this feature can be used to reduce the computational effort. First, we will show that the CR methods we discussed both maintain symmetry through each cycle of the cyclic reduction approach.

For the traditional cyclic reduction method, assume that at cycle $(k-1)$, $U_i^{(k-1)} = (L_{i+1}^{(k-1)})^T$ and $D_i^{(k-1)}$ is symmetric, which is valid at the first cycle for the original KKT linear system. Considering the update formulas (5.11) and (5.12), it is easy to show $D_i^{(k)}$ is still symmetric and $U_i^{(k)} = (L_{i+1}^{(k)})^T$.

For Yalamov's cyclic reduction methods, assume at cycle $(k-1)$ that $U_{i+2^{k-1}}^{(k-1)} = (L_{i-2^{k-1}}^{(k-1)})^T$ and $D_i^{(k-1)}$ is symmetric. These conditions are valid at the first cycle for the original KKT linear system as well. Combining update (5.14) and (5.16), it is easy to show that the relationship is still valid at cycle (k).

Therefore in both methods we only need to update matrices L, which reduces computational cost of the update formulas. However, the number of factorizations required stays the same. In other words, taking advantage of symmetry won't affect the time complexity analysis.

Moreover, when the symmetric update is applied (with only L updated), the accumulated run-off error may cause D_i to become less symmetric with each pass, so we correct D_i by reassigning D_i to $(D_i + D_i^T)/2$ at each cycle in order to balance the error. Our computational results in section 5.5.2 demonstrate the effectiveness of this approach.

5.3.5 Singular Diagonal Blocks

In previous sections, we assumed that the D_i blocks to be nonsingular. Actually, the first block D_1 is naturally degenerate because z_1 is not a free variable. This means that the KKT

matrix corresponding to the first diagonal block has more constraints than variables. In traditional cyclic reduction, we can pivot on the even diagonals (the lower right section of the matrix in (5.9)) to eliminate the upper right section of the matrix. Under this modified version of algorithm from Section 5.3.1, we no longer require the inverse of D_1.

Besides D_1, the other D_i could be singular as well. In general in dynamic optimization W_i is nonsingular. Since we assume the inertia of the whole KKT matrix is $(n, m, 0)$, the algebraic constraint Jacobian H_i within each D_i block is full rank as well. However, because part of the continuity constraints are in the off-diagonal blocks, there may also be a singularity of D_i arising from the differential constraint Jacobian $G_i(w_i)$.

To ensure that every D_i is nonsingular, we add slack variables s_i to all continuity equations. The new NLP is shown as follows,

$$\begin{aligned} \min \quad & \sum_{i=1}^{N-1} \hat{\Phi}(z_i, w_i) + \hat{\Psi}(z_N) + \sum_{i=1}^{N} \rho ||s_i||_2^2 \\ s.t. \quad & G_i(w_i) - z_{i+1} = s_i \;\; i = 1, \ldots, N-1 \\ & H_i(z_i, w_i) = 0, \;\; i = 1, \ldots, N \end{aligned} \tag{5.18}$$

Then the KKT matrix is modified as

$$\begin{bmatrix}
\begin{array}{|cccc|}\hline
W_{w_1w_1} & 0 & \nabla_{w_1}H_1 & \nabla_{w_1}G_1 \\
0 & \rho I & 0 & -I \\
\nabla_{w_1}H_1^T & 0 & 0 & 0 \\
\nabla_{w_1}G_1^T & -I & 0 & 0 \\ \hline
\end{array} & & & & \\
 & \begin{array}{|ccccc|}\hline
-I & & & & \\ \hline
\end{array} & & & \\
 & & \begin{array}{|ccccc|}\hline
W_{z_2z_2} & W_{z_2w_2} & 0 & \nabla_{z_2}H_2 & \nabla_{z_2}G_2 \\
W_{w_2z_2} & W_{w_2w_2} & 0 & \nabla_{w_2}H_2 & \nabla_{w_2}G_2 \\
0 & 0 & \rho I & 0 & -I \\
\nabla_{z_2}H_2^T & \nabla_{w_2}H_2^T & 0 & 0 & 0 \\
\nabla_{z_2}G_2^T & \nabla_{w_2}G_2^T & -I & 0 & 0 \\ \hline
\end{array} & & \\
 & & & \ddots & -I \\
 & & & -I & \begin{array}{|ccc|}\hline
W_{z_Nz_N} & W_{z_Nw_N} & (\nabla_{z_N}H_N \\
W_{w_Nz_N} & W_{w_Nw_N} & (\nabla_{w_N}H_N \\
\nabla_{z_N}H_N^T & \nabla_{w_N}H_N^T & 0 \\ \hline
\end{array}
\end{bmatrix} \tag{5.19}$$

and the individual D_i are now nonsingular. In our case study section 5.7, we applied these corrections whenever we apply CR methods.

5.4 Implementation

We implemented both traditional cyclic reduction and Yalamov cyclic reduction methods in Matlab 7.14.0.739 (R2012a) with the Matlab parallel computing toolbox. MA57 3.9.0 is called through MEX interface as a symmetric linear solver inside cyclic reduction methods and also is used directly as a serial method for comparison. All results are obtained on an Intel(R) Xeon(R) CPU E5-2440 @ 2.40GHz×12 with 64 GiB memory and 8 cores are used for all the experiments.

In all of our experiments, Matlab's built-in linear solver which is invoked by a backslash operator is used as a benchmark solver. We refer to this as 'Backslash' in the following sections. Backslash uses a well-tuned set of heuristics to determine which algorithm to use for

the linear system. First, it evaluates the structure of the linear system, including checking for symmetry, sparsity, or block diagonal structure. Then it selects the corresponding state-of-art linear solver such as UMFPACK[64], MA57[31] or LAPACK's banded solver[65]. We choose Matlab Backslash as a benchmark solver because of its well known high performance.

5.5 Testing Results for Randomly Generated Linear KKT Systems

In this section, six different linear solvers are compared on a test set of randomly generated linear systems. Ten randomly symmetric tridiagonal linear systems are generated for various choices of block size and number of blocks. The diagonal blocks are obtained by randomly generating a matrix with 3% density, then adding its transpose to guarantee symmetry. The off-diagonal matrices are random matrices with 3% density. Matlab function *sprand(m,n,den)* is applied for matrix generations. It generates a random, m-by-n, sparse matrix with approximately $den \times m \times n$ uniformly distributed nonzero entries in the range $(0, 1)$. We compare the CPU time and rhs errors among these six methods where the rhs error ϵ is defined by $||Ax - b||_\infty$. To provide stable timings, we solved all linear systems three times and the average CPU time was used for comparison.

The six linear solvers under consideration are:

1. *MA57*: Using Matlab to call MA57 through MEX interface
2. *Backslash*: mldivide(A,B) in Matlab
3. *TCR*: Traditional cyclic reduction method without considering symmetry
4. *TCR with symmetry*: Traditional cyclic reduction method using symmetric updates
5. *YCR*: Yalamov's cyclic reduction method without considering symmetry

6. *YCR with symmetry*: Yalamov's cyclic reduction method using symmetric updates

5.5.1 Rhs Error and CPU time variation

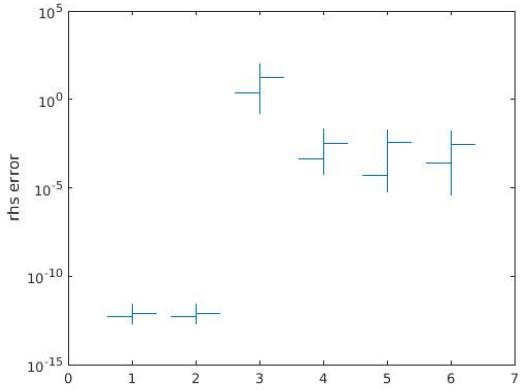

Figure 5.1: rhs error log scale

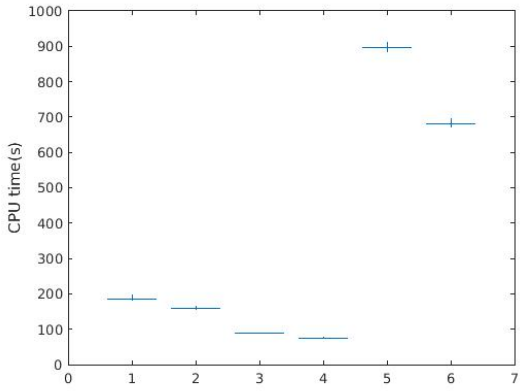

Figure 5.2: cpu time linear scale

In the first comparison, we fix the number of blocks at 500 ($N = 500$) and fix the block size at 500×500. Fig. 5.1 and Fig. 5.2 are stock plots for CPU time and rhs error (ϵ). Each figure has six vertical lines corresponding to each algorithm. The algorithms are labeled according to the numbering above. The vertical line runs from the minimum and maximum of these values among all 10 test problems. The mean values are marked as a horizontal line on the right side while median values are marked on the left side.

Note that the rhs errors from *MA57* and *Backslash* are the same for all 10 test cases, so we conclude that Matlab Backslash is most likely using *MA57* as the symmetric linear system solver for these systems. However, from Fig. 5.2, we can see using *Backslash* is slighter faster than applying *MA57* directly even if it takes time for Matlab to recognize it as a symmetric matrix. This is expected because *Backslash*'s interface to *MA57* is more memory efficient than our MEX interface.

Compared to YCR, TCR is less accurate. However, when including the symmetric correction formula for the KKT matrices, traditional CR has a very similar level of rhs error as Yalamov's method.

In Fig. 5.2, Yalamov's methods are much slower than the other methods since it is an $O(N \log N)$ method as we discussed in section 5.3.3. As expected, the symmetry-exploiting variants are generally faster than the base algorithms. Also, traditional CR methods are faster than *Backslash* even if there is a memory allocation disadvantage when using the MEX interface for the factorization of blocks within the CR method.

5.5.2 Increasing Number of Blocks

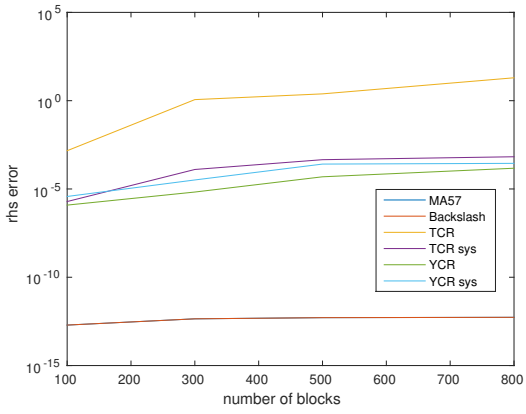

Figure 5.3: rhs error log scale

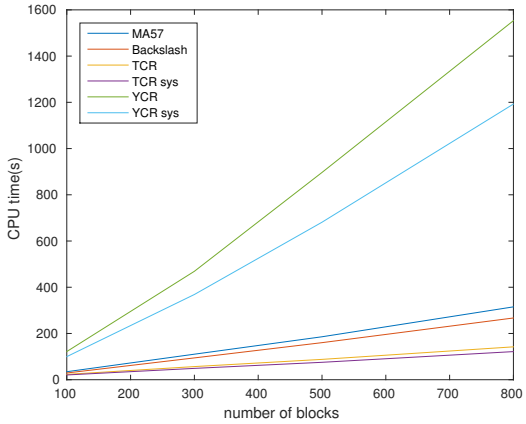

Figure 5.4: cpu time linear scale

In this section, we compare the scaling of the methods with the number of blocks. We fix the block size at 500×500 and vary the number of blocks (N) from 100 to 800. The median rhs error and CPU times among 10 randomly generated test cases are used for comparison. In Fig. 5.3, the median rhs error is plotted on a log scale against varying number of blocks. The traditional cyclic reduction method is not as accurate, with rhs error increasing as the

number of blocks increases. However, for the tradition cyclic reduction with symmetry and Yalamov's cyclic reduction methods, the rhs errors are all near 10^{-5} and don't change as strongly as the number of blocks changes. Matlab Backslash and MA57 have the best error control, as expected. Note that these lines overlap exactly because the error is the same. Because MA57 can choose pivots from the entire matrix, it has more flexibility to control the error. Cyclic reduction, however, can only choose pivots within the blocks under consideration.

In Fig. 5.4, the median CPU time is used for each number of blocks as an indicator of efficiency. In general, the symmetric methods perform better than the non-symmetric methods, since the symmetric methods require fewer operation counts. However, this is not a major improvement because the performance bottleneck of all algorithms is the factorization of the KKT matrix, and the symmetric methods do not impact the numbers of factorizations. Traditional CR methods perform better than Yalamov's methods, which is consistent with the results of time complexity analysis. In fact, the computational results match very well to a plot of $O(N)$ vs $O(N \log N)$ time complexity. As a result, the performance difference between the methods is amplified by the number of blocks. The more blocks we have, the larger the performance gap. This suggests that traditional CR methods will have superior speed for large scale problems.

5.5.3 Increasing Block Size

In this section, we fix the number of blocks at 500 and vary the size of blocks from 100×100 to 800×800. In Fig. 5.5, again we see that traditional cyclic reduction is unstable, and the rhs error increases as the size of blocks increases. However, for Yalamov's cyclic reduction methods and traditional cyclic reduction with symmetry, the rhs error doesn't vary much with the size of blocks. Compared to Fig. 5.4, the trend in Fig. 5.6 is more severe since

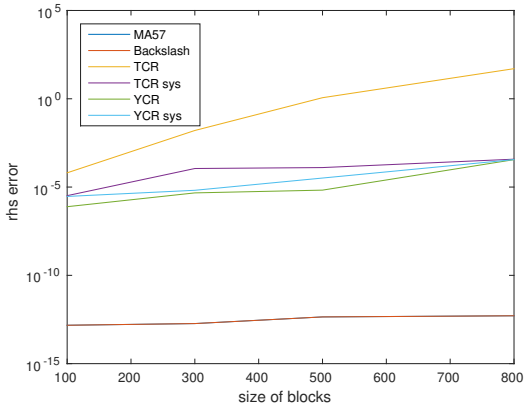

Figure 5.5: rhs error log scale

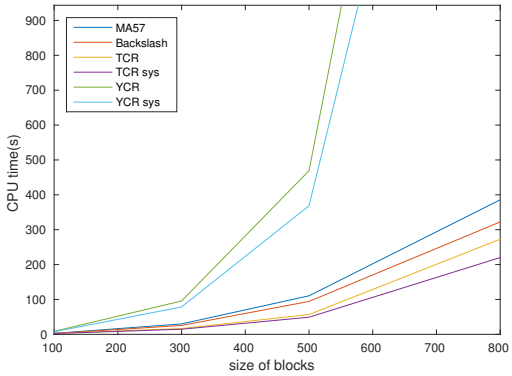

Figure 5.6: CPU time linear scale

the total number of nonzeros increases quadratically with the size of blocks. However, the conclusion remains the same: the Yalamov CR method is slower than traditional CR or standard linear solvers, and traditional CR with symmetry is the fastest method.

5.6 Linear System Testing With Dynamic Structure

In the previous section, we compared all six block tridiagonal linear system solvers using randomly generated linear systems. The computational results show that the traditional cyclic reduction method that utilizes symmetry is the best linear solver considering both CPU time and rhs error. In this section, we wish to further examine the performance of this method on random linear systems that even more closely represent the sparsity structure encountered in dynamic optimization problems. Three methods, *Backslash*, *MA57*, and *TCR with symmetry*, are tested on a new set of linear systems with dynamic optimization structure.

After permuting into block tridiagonal structure, the individual blocks of the KKT matrix have a specific sparsity structure, as discussed in section 5.2.2. The sparsity structure

can be described in terms of the size of the problem (5.1) and the discretization scheme used. First, define n_z as the number of states, n_y as the number of algebraic variables, and n_u as the number of control variables in problem (5.1). Now, consider the transformation of discretization (5.2) into form (5.3). If discretization uses K collocation points, then the aggregate variable vector $w_i^T = [z_{i,j}^T, z_{i,j}^T, y_{i,j}^T, u_{i,j}^T]$ is of dimension

$$n_w = K(2n_z + n_y + n_u).$$

The remaining variables in (5.3) are the state variables at finite element boundaries, z_i.

As shown in (5.7), each block diagonal matrix D_i is in a KKT form, with an upper left "Hessian" part, and the upper right/lower left "Jacobian" part. Define

$$n_n := n_w + n_z$$

and

$$n_m := n_w + n_z - Kn_u.$$

Except for the first and last blocks, the Hessian part of D_i is of dimension $n_n \times n_n$, while the Jacobian part (upper right sub-block) is of dimension $n_n \times n_m$. The off diagonal identity matrices in (5.3) are all of dimension n_z.

In this section, the linear systems are randomly generated based the pattern of (5.7) except that all blocks are the same size. The random generator is the same with the one we mentioned in section 5.5. For each D_i, a random matrix $X \in \mathbb{R}^{n_n \times n_n}$ is generated with density den and the Hessian part of D_i is set to $X + X^T$ to guarantee the symmetry. Unless otherwise specified, $den = 0.01$ in this section. The Jacobian part of D_i is generated with density den as well. Finally, the off-diagonal identity matrices are determined based on the specified value of n_z. As in section 5.5, 10 linear systems are generated for each size and the median of these 10 cases is used as an average performance. We vary the number

of continuity constraints n_z, the ratio of variables and constraints in each block, and the density den to compare how the performance varies for *Backslash*, *MA57*, and *TCR with symmetry*.

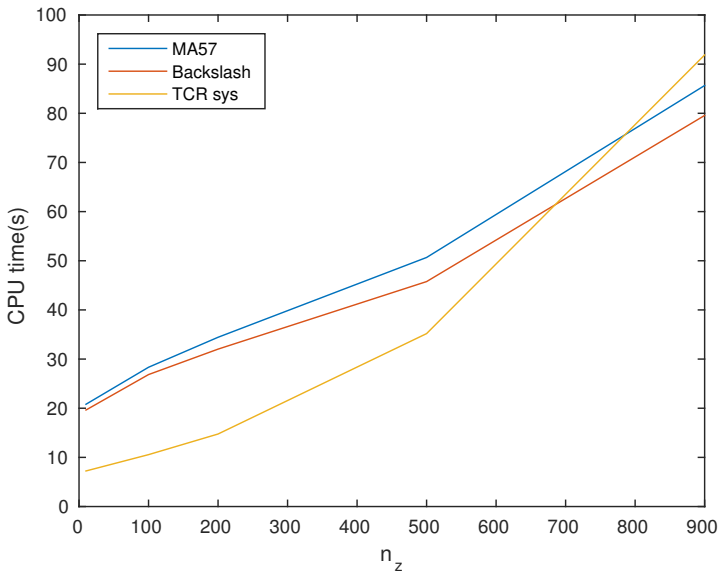

Figure 5.7: CPU time versus n_z

5.6.1 Increasing Number of Connection Variables

We set number of finite elements to $N = 20$, $n_n = 1000$, $n_m = 900$ and we vary n_z from 10 to 900. The CPU time performance is shown in Fig. 5.7. Because of the special (highly sparse) structure of the off-diagonal blocks, CR performs better compared to the serial methods than linear systems of similar size from section 5.5.

As shown in [59], the cyclic reduction maintains the sparsity of the off-diagonal matrices. Therefore in the implementation all zero columns are ignored when we calculate the auxiliary matrices P_i and Q_i in (5.11). This means that increasing the dimension of z_i increases the number of backsolves for update (5.11). In addition, the MA57 Matlab inter-

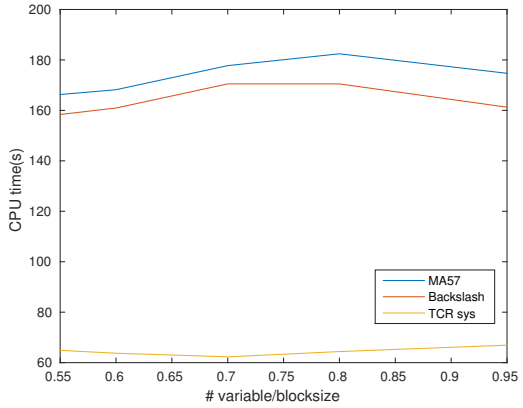

Figure 5.8: CPU time versus variable ratio

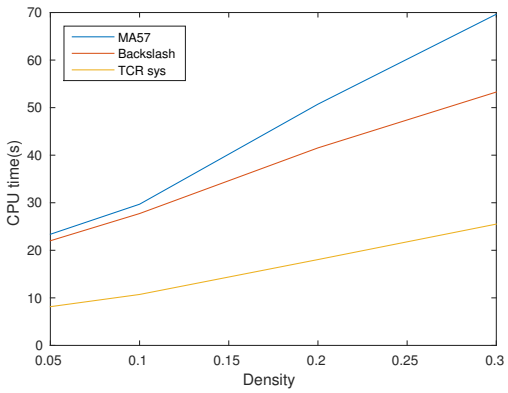

Figure 5.9: CPU time versus density

face only accepts the $L_{2i-1}^{(k-1)}$ matrix in full format rather than sparse, which increases the data operation time. Since this interface is used repeatedly in the TCR symmetry method, this contributes to the additional deterioration in performance for this method at large n_z. For all methods, increasing n_z leads to more fill-ins in the matrices and therefore longer computation times.

In Fig 5.7, *TCR symmetry* is better than *Backslash* and *MA57* for smaller n_z, but it gets much worse as n_z increases. However, in section 5.5 we showed that *TCR symmetry* can be better than *Backslash* and *MA57* even for random off-diagonal matrices, but at a much larger scale. In practice, the continuity equations are usually only a small fraction of all the constraints, so n_z is normally small.

5.6.2 Ratio of variables and constraints with fixed block size

We now set $N = 20$, the blocksize of D_i to 5000 and vary the ratio $\frac{n_n}{n_n+n_m}$ from 0.55 to 0.95. The CPU time performance is shown in Fig. 5.8. The performance stays roughly the same as the ratio of variables and constraints in D_i varies. Compared to the Schur complement

methods introduced in [59], this is a significant advantage for cyclic reduction methods. The parallel algorithm is not sensitive to the number of control and algebraic variables, which allows for application to a wider variety of dynamic optimization problems.

5.6.3 Density of block diagonal matrices

We now set $N = 20$, $n_n = 1000$, $n_m = 900$, $n_z = 10$ and vary the block diagonal matrix density den from 0.05 to 0.3. The CPU time performance is shown in Fig. 5.9. It is clear that the CPU time increases as the density increases since the computational expense of all of our algorithms is related to the number of fill-ins. As the density increases, the performance gap between *MA57* and *Backslash* gets larger, which is explained by the memory efficiency of Matlab's interface for MA57 compared to our interface and the larger amount data to transfer. On the other hand, for large dynamic optimization problems the density is often less than 0.01.

5.6.4 Parallel performance

To further understand the parallel performance of cyclic reduction, the computation time spent calling the external linear solver was recorded. In Fig. 5.10, time spent calling the external linear solver is highlighted in blue for one linear solve with TCR with symmetry. The KKT system used for this plot has $N = 64$ diagonal blocks with $n_n = 1000$, $n_m = 900$, and $n_z = 10$. The horizontal axis is the timeline from $t = 0$ to the time the solution is obtained at $t = 15.33$ seconds. The vertical axis labels each of the eight processors working on the problem. The blue blocks show the time period that is spent on MA57 external calls. If two calls to MA57 are very close in sequence, we have slightly expanded the gap to make the distinction more clear.

It is clear that external calls to solve linear systems constitute the major computational

expense of this algorithm, further justifying the assumption in complexity analysis. However, the distribution between the 8 processors is fairly even. We can clearly see that each cycle of cyclic reduction reduces the number of linear solves by a factor of two. The first cycle lasts about 5.5 seconds. In this cycle, 32 factorizations are done corresponding to 32 diagonal blocks. Similarly, the second cycle (ending around $t = 9$) and the third cycle (ending around $t = 10.5$) contain 16 and 8 external calls respectively. From the fourth cycle onward, the processors are not fully loaded due to fewer factorizations required.

As we know from section 5.6.3, calling MA57 from Matlab's MEX interface has a larger overhead than using the interface through Backslash. With a more efficient interface, every blue bar in Fig. 5.10 could be shortened, leading to better performance of the cyclic reduction method.

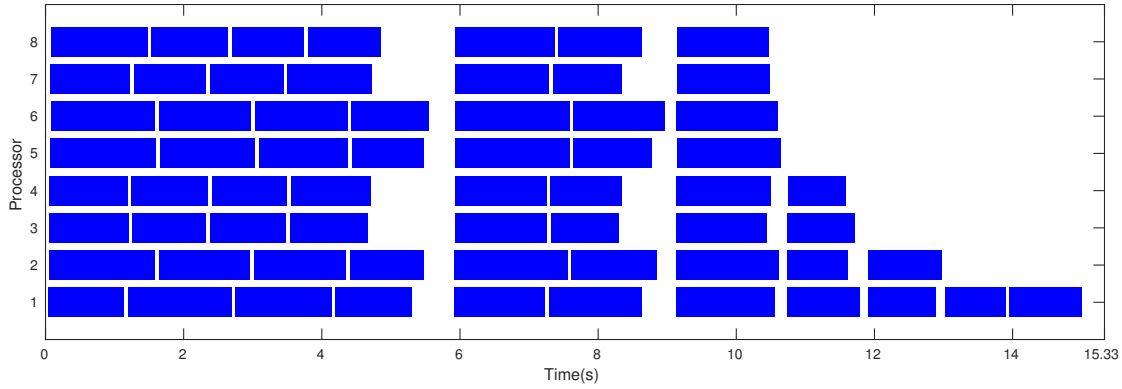

Figure 5.10: MA57 calls on each processor

5.7 Dynamic Process Optimization Examples

We now apply both TCR with symmetry and Matlab Backslash as linear solvers to solve four dynamic process optimization problems. These tests will allow us first to determine

whether the overall NLP algorithm can tolerate the relatively larger rhs error. Then, we can evaluate the potential for parallel speedup by using CR methods.

The four case study models are built in the AMPL [66] modeling environment and solved by an IPOPT-like solver in MATLAB (i.e. MIPOPT). MIPOPT uses ASL to perform function and derivative evaluations. Since cyclic reduction methods require a special structure in the KKT matrices, we pre-process the model to determine the permutation order. Because the NLP solver requires primal and dual variables to be sorted in the default order (all primal first, then dual), the permutation is applied before each cyclic reduction solve, then reversed afterwards. This assures that the NLP algorithm applied in the parallel CR method and the serial method are the same except for the choice of linear solver. However, this increases the computational expense in the cyclic reduction approach. Therefore, we use cycle count and average KKT solve time as performance metrics instead of total CPU time.

5.7.1 Auto permutation tool

Typically, when a NLP solver needs to solve a linear KKT system, the primal and dual variables are in default order (5.5). This means that the KKT matrix, from the perspective of the NLP solver, is always in form (5.4). To directly integrate our CR methods without modifying the NLP solver, we need to permute the matrix into block tridiagonal form at each cycle, then permute the system back to return it to the optimization method. This means we permute the original KKT system into variable order (5.6) and matrix structure (5.7). In this section, we introduce an auto permutation tool that we developed to automatically generate the permutation matrix directly from an AMPL model. Once we know the permutation matrix, it is easy to permute the KKT matrix to a block tridiagonal matrix for cyclic reduction methods and permute it back for the general IPOPT routine.

To make the permutation easier, we require that all state variable names in AMPL have a prefix "s_" and all continuity constraint names have a prefix "con_". Also we require that when variables and constraints are indexed over finite elements and collocation points, the finite element index comes first.

In the pre-processing phase, we read the AMPL auxiliary files (.col, .row). These files give the AMPL default order for constraints and variables. Using both of these lists, we determine the new ordering of variables and constraints that gives the block tridiagonal structure. The permutation orderings are stored in memory and used at each cycle.

5.7.2 MHE-CSTR

The first case study is state-estimation of a continuous stirred tank reactor. The dynamic optimization problem represents the estimation at a single time point within a moving horizon estimation framework.

The dynamic model, taken from Qu and Hahn [67], describes the exothermic reaction between sodium thiosulfate (component A) and hydrogen peroxide (component B).

$$\frac{dC_A}{dt} = \frac{F}{V}(C_A^{in} - C_A) - 2k(T_R)C_A^2 \tag{5.20a}$$

$$\frac{dT_R}{dt} = \frac{F}{V}(T_R^{in} - T_R) + \frac{2(-\Delta H_R)k(T_R)C_A^2}{\rho C_P} - \frac{UA}{V_\rho C_P}(T_R - T_{CW}) \tag{5.20b}$$

$$\frac{T_{CW}}{dt} = \frac{F_{CW}}{V_{CW}}(T_{CW}^{in} - T_{CW}) + \frac{UA}{V_{CW}\rho_{CW}C_{P_{CW}}}(T_R - T_{CW}) \tag{5.20c}$$

This model was used to estimate all three state variables, concentration C_A, reactor temperature T_R, and cooling water temperature T_{cw}. The estimation uses noisy measurements of two states, T_R and T_{cw}. The values of control variables F and F_{cw} are known. Parameters C_A^{in}, T_R^{in}, T_{cw}^{in}, V, A, ΔH_R, ρ, C_p, and U were taken from Rajaraman et al. [68].

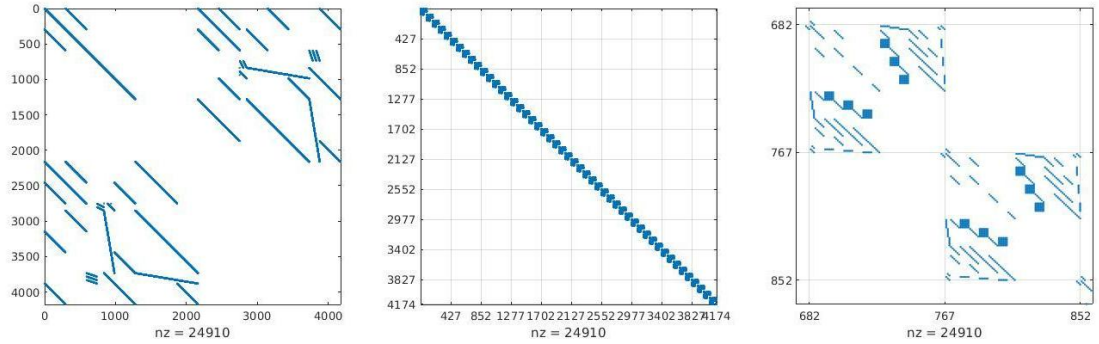

Figure 5.11: KKT sparsity patterns for MHE-CSTR

The left graphic of Fig. 5.11 presents the original KKT sparsity pattern, which follows the default order of variables and constraints from the AMPL ASL interface. The Hessian and Jacobian parts of the KKT matrix are clearly distinguishable. The KKT matrix after the re-ordering of variables and constraints based on the finite elements is shown in the middle graphic of Fig. 5.11. There are 50 small blocks on the diagonal, corresponding to 50 finite elements ($N = 50$). These blocks appear to be dense, but if we zoom in to see two finite elements, the symmetric sparsity pattern in each block is shown in the right graphic of Fig. 5.11. Note that there are three continuity constraints in the off-diagonal blocks to connect the variables. Within the diagonal blocks, we can again see the symmetric KKT structure. Within the Jacobian part, there are three small dense blocks resulting from the collocation equations. This example uses six collocation points per finite element.

The problem was solved by MIPOPT using several linear solvers, and the results are shown in Table 5.1. All problems converged in 22 cycles and the KKT matrix is solved within one second. The lack of parallel improvement is due to the relatively small problem size. However, we note that the additional rhs error introduced by TCR symmetry does not change the number of cycles required in the NLP solver. This means that the rhs error is manageable.

Table 5.1: MHE CSTR results

	MA57	Backslash	TCR symmetry
#Iter	22	22	22
Aver KKT solve time	0.0309	0.028	0.806

5.7.3 NMPC-CSTR

The second case study is nonlinear model predictive control of a CSTR benchmark problem presented in [69]. The dynamics of the CSTR are described by the following differential equations:

$$\frac{dc_A}{dt} = F(c_{A0} - c_A) - k_1 c_A - k_3 c_A^2 \tag{5.21a}$$

$$\frac{dc_B}{dt} = -F c_B + k_1 c_A - k_2 c_B \tag{5.21b}$$

$$\frac{dT_R}{dt} = F(T_{in} - T_R) + \frac{k_W A}{\rho c_p V_R}(T_K - T_R) - \frac{k_1 c_A \Delta H_{AB} + k_2 c_B \Delta H_{BC} + k_3 c_A^2 \Delta H_{AD}}{\rho c_p} \tag{5.21c}$$

$$\frac{dT_K}{dt} = \frac{1}{m_K c_{pK}}(\dot{Q}_K + k_W A(T_R - T_K)) \tag{5.21d}$$

where the reaction rates k_i follow the Arrhenius law:

$$k_i = k_{0,i} e^{\frac{-E_{A,i}}{R(T_R + 273.15)}} \tag{5.22}$$

The ODEs are derived from component balances for the species A (with concentration c_A) and for species B (with concentration c_B). The reactor temperature (T_R) and the coolant temperature (T_K) evolve in time according to the energy balance, which considers the flow in and out of the reactor, cooling duty, and reaction rates. The control inputs are the inflow (F) normalized by the volume of the reactor and the heat removed by the coolant ($\dot{Q}_K$).

The parameters that appear in the model equations are described in Table 5.2. The initial conditions and constraints on the states are described in Table 5.3, while the constraints for the control inputs are shown in Table 5.4.

Table 5.2: Parameter values of the NMPC CSTR

Parameter	Value	Unit	Parameter	Value	Unit
$k_{0,1}$	1.287×10^{12}	h^{-1}	ρ	0.9342	kg/l
$k_{0,2}$	1.287×10^{12}	h^{-1}	c_p	3.01	$kJ/(kg \cdot K)$
$k_{0,3}$	9.043×10^{9}	$l/(mol \cdot h)$	c_{pK}	2.0	$kJ/(kg \cdot K)$
$E_{A,1}/R$	9758.3	K	A	0.215	m^2
$E_{A,2}/R$	9758.3	K	V_R	10.0	l
$E_{A,3}/R$	8560.0	K	m_k	5.0	kg
ΔH_{AB}	4.2	kJ/mol	T_{in}	130.0	°C
ΔH_{BC}	-11.0	kJ/mol	k_W	4032	$kJ/(h \cdot m^2 \cdot K)$
ΔH_{AD}	-41.85	kJ/mol			

The left graphic of Fig. 5.12 is the original KKT matrix sparsity pattern, while the middle one shows the sparsity pattern after rearrangement. This example uses 40 finite elements ($N = 40$) with 3 collocation points each. Because of fewer collocation points, the block size is slightly smaller than in the previous case study. In the right graphic of Fig. 5.12, the collocation constraints for each of the 4 state variables are visible, and the continuity constraints are observed in the off-diagonal blocks.

The problem is solved in MIPOPT with MA57, Matlab Backslash, and traditional cyclic reduction with symmetry. The results are presented in Table 5.5. All solves converge quickly in a reasonable number of cycles, and the average KKT solve time is very small.

Table 5.3: State variables of the NMPC CSTR

State	Initial Condition	Lower Bound	Upper Bound	Unit
c_A	0.8	0.1	5.0	mol/l
c_A	0.8	0.1	5.0	mol/l
T_R	134.14	50.0	180.0	°C
T_K	134.0	50.0	180.0	°C

Table 5.4: Control variables of the NMPC CSTR

Control	Min.	Max.	Unit
F	5	100	h^{-1}
$\dot{Q}_K$	-8500	0	kJ/h

Again, parallel improvement is not observed because of the small problem size. This means that the expense of calling the external linear solver within CR does not dominate the other costs of the parallel algorithm, such as data transfer and parallelization set-up.

5.7.4 Polymer grade transition FBR

The third case study comes from operation of polymerization reactors. A single polymerization reactor is used to produce many different grades of a polymer. When operating this reactor, it is common to switch between grades without shutting down the process. Any product produced during the transition is considered off-grade, usually with far less value. This case study is a dynamic optimization problem to minimize the production of

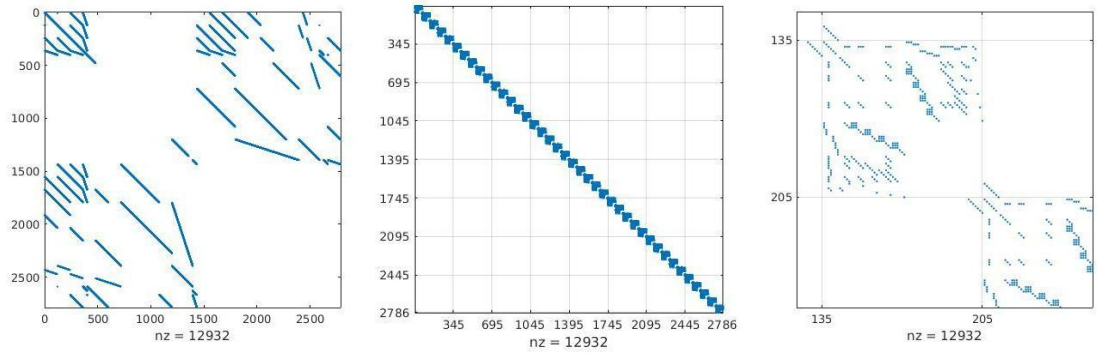

Figure 5.12: KKT sparsity patterns for NMPC CSTR

Table 5.5: NMPC CSTR results

	MA57	Backslash	TCR symmetry
#Iter	24	24	26
Aver KKT solve time	0.0153	0.0138	0.754

this off-specification product.

The model considers the copolymerization of ethylene and propylene. The reaction system is described by the set of reactions shown in Table 5.6. C_p represents a potential catalyst site, P_0 represents an activated catalyst site, C_d represents dead site, A is the cocatalyst, M_i represents monomer species $i \in \{C_3H_6, C_2H_4\}$, $P_{n,i}$ represents a live polymer with chain length n and end group i, and $D_{n,i}$ represents a dead polymer with chain length n and end group i.

The polymer system is modeled using the method of moments to decrease the number of states. Instead of modeling each chain length of polymer as a separate state variable, the first three moments of the molecular weight distribution are used as states instead. The

Table 5.6: Polymer grade transition FBR reactions

Reaction step	Reactant	Reaction	Rate constant
Site activation	Hydrogen	$C_p + H_2 \rightarrow P_0$	k_{AH}
	Cocatalyst	$C_p + A \rightarrow P_0 + B$	k_{AA}
Chain initiation	Monomer i	$P_0 + M_i \rightarrow P_{1,i}$	k_{Ii}
Chain propagation	Monomer j	$P_{n,i} + M_j \rightarrow P_{n+1,j}$	k_{Pj}
Chain transfer	Hydrogen	$P_{n,i} + H_2 \rightarrow P_0 + D_{n,i}$	k_{TH}
	Monomer j	$P_{n,i} + M_j \rightarrow P_{1,j} + D_{n,i}$	k_{TMj}
	Spontaneous	$P_{n,i} \rightarrow P_0 + D_{n,i}$	k_{TS}
Site deactivation	Spontaneous	$P_{n,i} \rightarrow C_d + D_{n,i}$ $P_0 \rightarrow C_d$	k_{DS}

differential equations are derived from the detailed reaction kinetics.

The k^{th} moment of the living polymer distribution is given by

$$\mu_k = \sum_{n=1}^{\infty} n^k \left[P_n\right]$$

and the k^{th} moment of the bulk polymer is given by

$$\lambda_k = \sum_{n=1}^{\infty} n^k ([D_n] + [P_n])$$

After applying the method of moments to describe moments $k = 0, 1$, and 2, the reaction system is described by six differential equations in Table 5.7.

The reactor includes a bubble phase and an emulsion phase. Solid catalyst, cocatalyst, and polymer are suspended in the emulsion phase. The bubble phase contains no solids, thus no reactions occur there. The gas component concentrations are represented by Y_i,

Table 5.7: Polymer grade transition FBR moments model

Moments	Reaction Rate
Zero order	$R_{\mu_0} = \sum_{i=1}^{NM}(k_{Ii}[P_0][M_i]) - ((k_{TH}+k_{DH})[H_2] + (k_{TS}+k_{DH}))\mu_0$
	$R_{\lambda_0} = \sum_{i=1}^{NM}(k_{Ii}[P_0][M_i])$
First order	$R_{\mu_1} = \sum_{i=1}^{NM}(k_{Ii}[P_0][M_i] + k_{Pi}[M_i]\mu_0 + k_{TMi}[M_i](\mu_0-\mu_1))$ $-((k_{TH}+k_{DH})[H_2] + (k_{TS}+k_{DS}))\mu_1$
	$R_{\lambda_1} = \sum_{i=1}^{NM}(k_{Ii}[P_0][M_i] + k_{Pi}[M_i]\mu_0 + k_{TMi}[M_i]\mu_0)$
Second order	$R_{\mu_2} = \sum_{i=1}^{NM}(k_{Ii}[P_0][M_i] + 2k_{Pi}[M_i]\mu_1 + k_{TMi}[M_i](\mu_0-\mu_2))$ $-((k_{TH}+k_{DH})[H_2] + (k_{TS}+k_{DS}))\mu_2$
	$R_{\lambda_2} = \sum_{i=1}^{NM}(k_{Ii}[P_0][M_i] + 2k_{Pi}[M_i]\mu_1 + k_{TMi}[M_i]\mu_0)$

for components $i \in \{H_2, C_3H_6, C_2H_4, C_3H_8, C_2H_6, N_2\}$. The solid particle concentrations are represented as S_j for solid components $j \in \{$catalyst, cocatalyst, polymer moments$\}$. The overall mass and energy balances are shown in Table 5.8, where parameters such as minimum fluidization velocity U_{mf}, bubble fraction δ, voidage ϵ_{mf}, mass transfer coefficient K_{be}, and heat transfer coefficient H_{be} are determined by semi-empirical correlations. These correlations, taken from [70, 71], are included as algebraic equations in the model.

After applying the method of moments, the system consists of 27 state variables, including concentrations of monomers, catalysts, inerts, temperature, and the moments of the molecular weight distribution. Due to the highly non-linear nature of the process, the differential equations are discretized with Radau collocation on 80 finite elements ($N = 80$).

Table 5.8: Mass balance and energy balance

Emulsion Phase

$$\frac{d[Y_i]_e}{dt} = \frac{U_e}{H}([Y_i]_{in} - [Y_i]_e) + \frac{(1-\delta)K_{be}}{\delta\epsilon_{mf}}([Y_i]_b - [Y_i]_e) + \frac{1-\epsilon_{mf}}{\epsilon_{mf}}R_i - \frac{[Y_i]_e}{H}\frac{dH}{dt}$$

$$\frac{d[S_j]}{dt} = \frac{Q_{in,j}}{V_e(1-\epsilon_{mf})}[S_j]_{in} - \frac{Q_{out}}{V_e}[S_j] + R_j - \frac{[S_j]}{H}\frac{dH}{dt}$$

$$\frac{dT_e}{dt} = \frac{U_e A_e \epsilon_{mf} \sum_i [Y_i]_{in} \int_{T_e}^{T_{in}} Cp_i dT + V_e(1-\epsilon_{mf})R_{pol}\Delta H_{pol} + V_b H_{be}(T_b - T_e) + V_b K_{be} \sum_i ([Y_i]_b - [Y_i]_e) \int_{T_e}^{T_b} Cp_i dT}{\sum_i V_e \epsilon_{mf}[Y_i]_e Cp_i + V_e(1-\epsilon_{mf})\rho_{pol} Cp_{pol}}$$

Bubble

$$\frac{d[Y_i]_b}{dt} = \frac{U_b}{H}([Y_i]_{in} - [Y_i]_b) - K_{be}([Y_i]_b - [Y_i]_e) - \frac{[Y_i]_b}{H}\frac{dH}{dt}$$

$$\frac{dT_b}{dt} = \frac{U_b A_b \sum_i [Y_i]_{in} \int_{T_b}^{T_{in}} Cp_i dT - V_b H_{be}(T_b - T_e)}{\sum_i V_b [Y_i]_b Cp_i}$$

Bed height

$$\frac{dH}{dt} = \frac{-Q_{out}(1-\delta)(1-\epsilon_{mf})\rho_{pol} - \sum_i (R_{M_i} M w_i) V_e (1-\epsilon_{mf})}{\rho_{pol} A_e (1-\epsilon_{mf})}$$

The objective minimizes the integral of squared deviation from the desired polymer properties, plus a regularization term on the controls, i.e.,

$$Min \quad \int_0^{T_f} \|z(t) - z^*\|_Q + \|u(t) - u^*\|_R dt. \tag{5.23}$$

In Fig. 5.13, the original KKT pattern looks relatively dense because most states interact with each other in the reactions. However, we can still permute this system based on the finite elements, as shown in the middle graphic of Fig. 5.13. If we zoom in to one finite element, the block is shown in the right graphic, with a relatively dense but well-defined structure.

This problem is slightly larger than the previous ones and more nonlinear because of the particular reaction kinetics. Therefore, optimization using all methods requires more

cycles to solve the NLP than the previous examples. Even though Matlab Backslash is still fastest in average KKT solve time, the gap between serial and parallel methods decreases with increasing problem size.

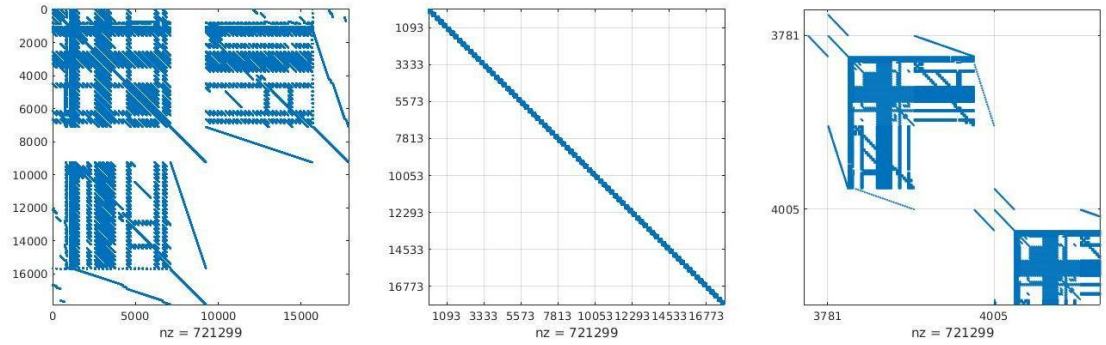

Figure 5.13: KKT sparsity patterns for polymer grade transition FBR

Table 5.9: Grade transition FBR results

	MA57	Backslash	TCR symmetry
#Iter	96	96	54
Aver KKT solve time	0.81	0.72s	0.88s

5.7.5 NMPC BFB

The final case study is motivated by post-combustion carbon capture. The system under consideration is a bubbling fluidized bed adsorber. Solid sorbent particles are added at the top of the bed while flue gas is added from the bottom of the bed. The solid and gas contact each other in counter-current flow and CO_2 adsorbs onto the surface of the solid particles. The solid particles are then transported to a regenerator, where increased

temperature causes the CO_2 to desorb for later compression and sequestration. In this case study, we solve one time step of nonlinear model predictive control for a detailed adsorber model.

This bubbling fluidized bed adsorber model developed by NETL considers adsorption kinetics and bed hydrodynamics to predict the axial and temporal variations in concentrations and temperature [72, 73]. The flow within the column is modeled using three flow regimes as shown in Figure 5.14: an upward flowing bubble region (containing no solids), an upward flowing cloud wake region (containing some solids), and a downward flowing emulsion region (containing solids). The adsorption reactions, which occur in the solid phase, are given as follows:

$$\begin{aligned} &\mathrm{H_2O_{(g)}} \rightleftharpoons \mathrm{H_2O_{(phys)}} \\ &2\,\mathrm{R_2NH^+CO_2} \rightleftharpoons \mathrm{R_2NH_2}^+ + \mathrm{R_2NCO_2}^- \\ &\mathrm{R_2NH + CO_2 + H_2O_{(phys)}} \rightleftharpoons \mathrm{R_2NH_2}^+ + \mathrm{HCO_3}^- \end{aligned} \tag{5.24}$$

Mass and energy balances are written for each of the three regions and each phase within those regions. The allowable set of mass and heat transfer flows are shown in Fig. 5.15. The total set of mass and energy balances consists of 20 partial differential equations. As an example, consider the gas phase mass balance equation in the bubble region:

$$\frac{\partial c_{b,j}}{\partial t}\delta A_x = -\frac{\partial G_b y_{b,j}}{\partial x} - A_x \delta K_{bc,j}(c_{b,j} - c_{c,j}) + K_{g,bulk,j} \tag{5.25}$$

The left hand side accounts for the accumulation of component j in the bubble region, where $c_{b,j}$ is the concentration of component j in the bubble region, δ is the volume fraction of the bubble region, and A_x is the cross-sectional area. The first right hand side represents the upward gas flow in the bubble region, where G_b is the axial flow rate and $y_{b,j}$ is the gas mole fraction for component j in the bubble region. The second term on the right hand side represents the mass transfer between the bubble and the cloud-wake regions, where

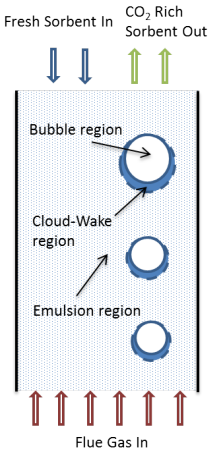

Figure 5.14: BFB adsorber

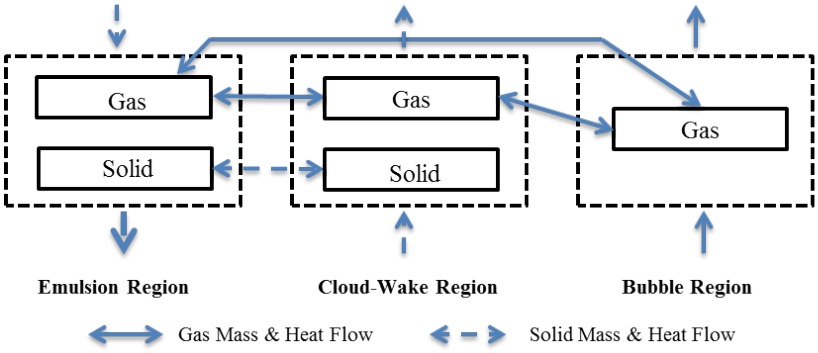

Figure 5.15: Mass and energy relations

$K_{bc,j}$ is the mass transfer coefficient and $c_{c,j}$ is the concentration of component j in the cloud-wake region. The final term $K_{g,bulk,j}$ represents the bulk flow of component j from the emulsion region into the bubble region. The value of this last term is determined by a separate algebraic equation. The other mass and energy balance PDEs may be found in [49, 72, 73].

The remainder of the model consists of algebraic equations. These equations include empirical correlations to describe the hydrodynamics, heat and mass transfer coefficient relations, gas phase properties including viscosity, thermal conductivity, and heat capacity, empirical correlations to describe the cooling tubes within the adsorber, and detailed nonlinear reaction kinetics for reactions (5.24).

When fixing the discretization in space, all the collocation variables in space become state variables in time. This leads to about 16000 variables and constraints for each finite element in time. We discretized the problem using 5, 10, and 16 time finite elements and compare the performance of the serial and parallel methods.

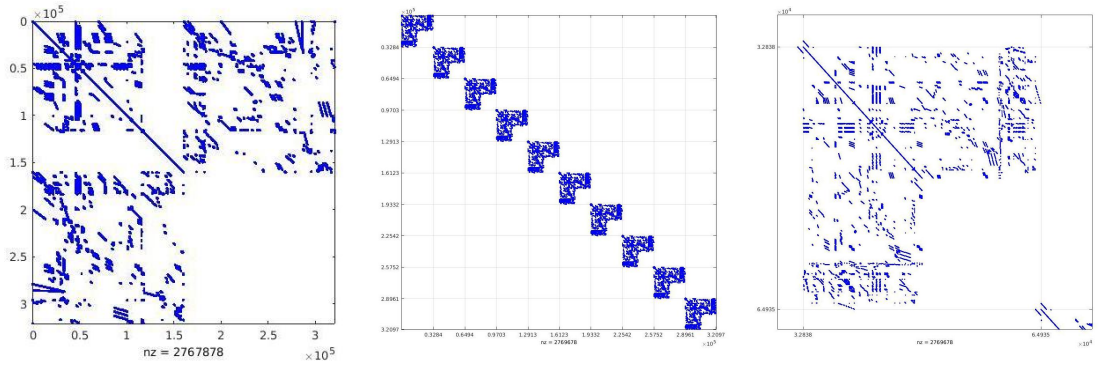

Figure 5.16: KKT sparsity patterns for NMPC BFB

In Fig. 5.16 we use 10 time finite elements ($N = 10$) as an example. This KKT matrix has 2.7 million nonzeros. The auto permutation tool permutes this system to block tridiagonal structure; the structure of one finite element is shown in the right graphic.

When using 16 finite elements, the number of variables and constraints is nearly 257000, and the order of the KKT matrix is about 514000. From Table 5.10, we can see that both methods converge the problem easily. Even though TCR with symmetry takes slightly more iterations than Matlab Backslash (due to larger rhs error discussed in section 5.5), parallelization significantly reduces the average KKT solve time for traditional cyclic reduction with symmetry as the problem size increased. With 16 finite elements, Matlab Backslash required 485.45 seconds for each linear system solve on average, while the traditional cyclic reduction with symmetry only takes 118.96 seconds.

5.8 Conclusions

In this chapter, we present two cyclic reduction methods to exploit the block tridiagonal structure in simultaneous dynamic optimization. Despite having better stability properties, Yalamov CR is no more accurate than traditional CR after exploiting symmetry. In

Table 5.10: BFB results

#Finite Elements	5	10	16
#Variables	80250	160500	256800
#Constraints	80235	160470	256752
MA57 #Iter	7	8	11
MA57 Aver KKT solve time	35.18s	189.39s	356.11s
Backslash #Iter	7	8	11
BS Aver KKT solve time	42.36s	182.63s	485.45s
TCR symmetry #Iter	13	14	17
TCR Aver KKT solve time	59.86s	98.29s	118.96s

addition, traditional CR is much faster than the Yalamov variant on a set of test matrices. The worst-case time complexity analysis explains the observed behavior very clearly.

In addition, we applied traditional CR with symmetry stabilization to 4 case studies from chemical engineering. The case studies show that an interior point method using our CR linear solver is competitive with the same method using serial linear solvers. The advantage of the parallel CR algorithm is most evident for the largest problem. With very many states and blocks, a parallel speedup of a factor of 4 was observed using 8 processors.

Future work will deal with a deeper analysis and refinement of the stability properties for CR methods. We also plan to implement these methods within an MPI environment in order to demonstrate these approaches on massively parallel machines.

Chapter 6

IPOPT for Mathematical Programs with Complementarity Constraints (MPCCs)

In this Chapter, we consider the solution of MPCCs with IPOPT. First, we will explore the ways to reformulate MPCCs to NLPs. Based on the reformulations, we apply two auto-adjustment penalty methods, then propose a constraint elimination method which can eliminate the dependent (equality or inequality) constraints locally. All methods are compared with MacMpec benchmark library and two large-scale applications are considered.

6.1 Introduction

Mathematical programs with complementarity constraints (MPCCs), are useful for optimization in many applications. Complementarity constraints require at least one of a pair of bounds to be active. This is useful for optimization since they can model certain disjunctions without binary variables. Although introducing binary variables and using mixed integer optimization is the more general approach to handling disjunctions, these solution methods may be computationally expensive for large nonlinear systems because worst-case complexity is exponential in the number of discrete decisions (NP hard). Complementarity constraints allow for solution techniques based on nonlinear programming, which can be used to quickly obtain local solutions.

A good review of MPCC applications is given in [74]. The authors review applications such as contact and friction mechanics, structural design, traffic equilibrium, optimal control, and market equilibria. In chemical engineering, applications include flow reversal, check valves, relief valves, controller saturation, and disappearance of phases in equilibrium calculations [75].

There is a long history of work on solution methods for MPCCs. Many papers use the term Mathematical Programs with Equilibrium Constraints (MPECs), which is a generalization of the complementarity constrained problem. The relation is outlined below in section 6.2. MPCCs or MPECs are difficult to solve because constraint qualifications do not hold. This is a challenge for nonlinear programming solvers. Many approaches have been proposed to transform the MPCC into an NLP that satisfy constraint qualifications, including relaxation methods [76, 77], smoothing methods [78], lifting methods [79, 80], and penalization methods [81, 82, 83].

In this work, we will focus on computational efficiency of MPCC solution methods. In Section 6.2, we will derive the MPCC formulation from a more general MPEC formulation, which itself is an example of how to reformulate a bilevel optimization into an MPCC. Then we discuss the features that make MPCC challenging and review some computationally attractive solution strategies. In Section 6.3, we will present two auto-adjusting penalty methods with convergence guarantees. In Section 6.4, we introduce a new constraint elimination method to eliminate the locally dependent constraints arising from complementarity constraints. The implementation details are provided in Section 6.5, and the algorithms are compared on the MacMPEC test set and results are presented in Section 6.6. In Section 6.7, we discuss two applications in chemical engineering, including building the MPCC model and testing with our proposed methods. Finally we conclude and discuss future directions.

6.2 MPCC Background

The general form of a MPCC we use in this chapter is given as follows:

$$\begin{aligned} min \quad & f(x) \\ s.t. \quad & c(x) = 0 \\ & g(x) \leq 0 \\ & 0 \leq y \perp z \geq 0 \end{aligned} \tag{6.1}$$

where $x = [w^T, y^T, z^T]^T \in \mathbb{R}^n$. Both complementarity variables y and z are in $\mathbb{R}^{n_c}$ and the remaining variables $w \in \mathbb{R}^{n-2n_c}$. The objective function is $f : \mathbb{R}^n \to \mathbb{R}$, equality constraints $c : \mathbb{R}^n \to \mathbb{R}^{m_E}$, inequality constraints $g : \mathbb{R}^n \to \mathbb{R}^{m_I}$. $\perp$ is the complementarity operator enforcing at least one of the bounds to be active. The complementarity constraint implies that

$$\begin{aligned} & y^{(i)} = 0 \text{ OR } z^{(i)} = 0, \ i = 1, \ldots, n_c, \\ & y \geq 0, \ z \geq 0 \end{aligned} \tag{6.2}$$

Here the OR operator is inclusive, as both variables may be zero.

MPCCs can be generalized as mathematical programs with equilibrium constraints (MPECs). A mathematical program with equilibrium constraints is an NLP with a variational inequality as a constraint, represented as follows:

$$\langle G(v_y), u - v_y \rangle \geq 0 \quad \forall u \in K$$

where $\langle \cdot, \cdot \rangle$ represents an inner product, and K is typically a convex set, and $v_y \in K$. If the functional G is a gradient mapping of some convex function g, i.e. $G(v_y) := \nabla g(v_y)$, and K is convex, then the variational inequality corresponds to the optimality conditions of a convex optimization problem. Assuming a finite dimensional variational inequality

we can include it in a mathematical program to form an MPEC, as follows:

$$\begin{aligned} \min_{v_x, v_y} \quad & f(v_x, v_y) \\ s.t. \quad & c(v_x, v_y) \geq 0 \\ & (u - v_y)^T G(v_x, v_y) \geq 0 \quad \forall u \in K(v_x) \end{aligned} \tag{6.3}$$

where we have introduced upper level variables v_x that may affect the set K. If the set K can be represented as a set of inequalities,

$$K(v_x) = \{v_y \mid s(v_x, v_y) \geq 0\}$$

Problem (6.3) can then be transformed to a MPCC as follows [81]:

$$\begin{aligned} \min_{v_x, v_y, \lambda} \quad & f(v_x, v_y) \\ s.t. \quad & c(v_x, v_y) \geq 0 \\ & G(v_x, v_y) - \nabla s(v_x, v_y)^T \lambda = 0 \\ & 0 \leq s(v_x, v_y) \perp \lambda \geq 0 \end{aligned} \tag{6.4}$$

Note that formulation (6.4) can be easily stated in the MPCC standard form (6.1). This is equivalent to using the KKT conditions when the variational inequality represents the optimality conditions of an inner convex optimization problems.

Now we will discuss how to go about solving MPCCs. Most solution approaches use nonlinear programming concepts. To apply existing NLP solution strategies, we can reformulate the complementarity constraints in (6.1) in several ways as follows,

$$\begin{cases} y^{(i)} z^{(i)} = 0, \ i = 1, \ldots, n_c \\ y^{(i)} z^{(i)} \leq 0, \ i = 1, \ldots, n_c \\ y^T z \leq 0 \\ y^T z = 0 \end{cases}, \qquad x \geq 0, y \geq 0 \tag{6.5}$$

The first two reformulations each contain n_c constraints while the last two reformulations only use one constraint to replace the complementarity constraints.

After reformulation, the MPCC problem becomes a general NLP problem. However, these NLP problems are hard to solve due to the nonsatisfaction of constraint qualifications. It is easy to show that neither LICQ nor MFCQ is satisfied at any feasible points for any of the reformulations in (6.5). For example, consider the first reformulation $y^{(i)}z^{(i)} = 0$. At a feasible point where $y^{(i)} = 0$, the constraint $y^{(i)}z^{(i)} = 0$ is dependent with the active bound $y \geq 0$, so LICQ does not hold. Because of the complementarity constraints, it is also clear that there is no feasible search direction into the interior of the inequalities ($y > 0$, $z > 0$), so MFCQ fails as well. Similar reasoning can be used to show that none of the reformulations satisfy LICQ or MFCQ. We will discuss this in more detail later in Section 6.4. As we mentioned in Chapter 2, without a CQ, an optimal solution may not be a KKT point, and NLP solvers designed to find KKT points may not be suitable. Therefore, we will first consider alternative optimality conditions for MPCCs.

6.2.1 MPCC optimality conditions

We focus on KKT-like optimality conditions for MPCCs. Depending on the assumptions, different stationarity conditions may be derived. However, the practicality of some of these constraint qualifications is debatable. The most desired stationarity condition is Bouligand stationarity, or B-stationarity. A feasible point w^* of (6.1) is said to be B-stationary if $d = 0$ is a solution of the linearized complementarity problem:

$$\begin{aligned} \min_d \quad & \nabla_x f(x^*)^T d \\ s.t. \quad & \nabla_x c(x^*)^T d = 0 \\ & g(x^*) + \nabla_x g(x^*)^T d \leq 0 \\ & 0 \leq y + d_y \perp z + d_z \geq 0 \end{aligned} \tag{6.6}$$

However, verification of B-stationarity would require solving an LPEC (linear program with equilibrium constraints). Therefore, most stationarity conditions propose some KKT-

like condition instead, which requires additional assumptions.

Consider the following KKT-like conditions for (6.1). We say that x^* is *weakly stationary* if there exist multipliers $(\lambda, \eta, \nu_y, \nu_z)$ such that

$$\begin{aligned} \nabla f(x^*) + \nabla c(x^*)^T \lambda + \nabla g(x^*)^T \eta - \begin{bmatrix} 0 \\ \nu_y \\ \nu_z \end{bmatrix} &= 0 \\ c(x^*) &= 0 \\ 0 \geq g(x^*) \perp \eta &\geq 0 \\ 0 \leq y^* \perp z^* &\geq 0 \\ y_i^* > 0 \implies \nu_{y,i} = 0, \text{ and } z_i^* > 0 \implies \nu_{z,i} &= 0, \ \forall i = 1, \ldots, n_c \end{aligned} \tag{6.7}$$

Note that a weakly stationary point still permits a descent direction if some $\nu_{y,i} < 0$ or $\nu_{z,i} < 0$ for some i. The condition may be strengthened in several ways. Based on the presentation in [84], we summarize these common stationarity conditions using the conditions they put on multipliers ν_y and ν_z. Define the set of biactive indices:

$$\mathcal{D}(x) := \{i \mid y_i = z_i = 0\} \tag{6.8}$$

1. A point x^* satisfying (6.7) is called *strongly stationary* if

$$\nu_{y,i} \geq 0 \text{ and } \nu_{z,i} \geq 0, \ \forall i \in \mathcal{D}(x^*) \tag{6.9}$$

2. A point x^* satisfying (6.7) is called *A-stationary* if

$$\nu_{y,i} \geq 0 \text{ or } \nu_{z,i} \geq 0, \ \forall i \in \mathcal{D}(x^*) \tag{6.10}$$

3. A point x^* satisfying (6.7) is called *C-stationary* if

$$\nu_{y,i}\nu_{z,i} \geq 0, \ \forall i \in \mathcal{D}(x^*) \tag{6.11}$$

4. A point x^* satisfying (6.7) is called *M-stationary* if

$$(\nu_{y,i} > 0 \text{ and } \nu_{z,i} > 0) \text{ or } \nu_{y,i}\nu_{z,i} = 0,\ \forall i \in \mathcal{D}(x^*) \tag{6.12}$$

Although each stationarity condition is valid with its corresponding assumptions, in practice only strong stationarity excludes the possibility of a trivial search direction on general problems. Therefore, we will examine in more detail the conditions associated with strong stationarity.

6.2.2 MPCC constraint qualification

Let $\bar{x} = [\bar{w}^T, \bar{y}^T, \bar{z}^T]^T$ be a feasible point of (6.1), and define the relaxed NLP (RNLP) as:

$$\begin{aligned} \min \quad & f(x) \\ s.t. \quad & c(x) = 0 \\ & g(x) \leq 0 \\ & y^{(i)} = 0,\ i \in I_Y/I_Z \\ & z^{(i)} = 0,\ i \in I_Z/I_Y \\ & y^{(i)} \geq 0, z^{(i)} \geq 0\ I_Y \cap I_Z \end{aligned} \tag{6.13}$$

where $I_Y = \{i : \bar{y}_i = 0\}$ and $I_Z = \{i : \bar{z}_i = 0\}$. If LICQ (see Chapter 2) holds for (6.13) at the point $\bar{x}$, then we say that MPCC-LICQ holds for (6.1) $\bar{x}$. MPCC-MFCQ is defined similarly.

It has been shown that strong stationarity implies B-stationarity [85]. With MPCC-LICQ, B-stationarity implies strong stationarity as well. The usefulness of strong stationarity as necessary optimality conditions is summarized with the following theorem:

Theorem 5. *[1, 85, 86] If $\bar{x}$ is a solution to the MPCC (6.1) and MPCC-LICQ holds at $\bar{x}$, then $\bar{x}$ is strongly stationary.*

6.2.3 Solving methods

Most solving methods for MPCCs of form (6.1) can be divided into two categories: NLP reformulations and specialized algorithms. The first category applies some transformation of the constraint (6.5) to convert the problem into normal NLP problem(s). Usually, this involves relaxation of the complementarity, and a sequence of NLPs are solved, to create a sequence that converges to the solution of (6.1). Here, we listed the most common relaxations for NLP tools to be applied. See [83, 86] for details on the convergence properties.

$$\begin{aligned} Reg(\epsilon): \quad \min \quad & f(x) \\ s.t. \quad & c(x) = 0,\ g(x) \leq 0 \\ & y, z \geq 0,\ y_i z_i \leq \epsilon,\ i = 1, \ldots, n_c \end{aligned} \tag{6.14}$$

$$\begin{aligned} RegComp(\epsilon): \quad \min \quad & f(x) \\ s.t. \quad & c(x) = 0, g(x) \leq 0 \\ & y, z \geq 0, y^T z \leq \epsilon \end{aligned} \tag{6.15}$$

$$\begin{aligned} RegEq(\epsilon): \quad \min \quad & f(x) \\ s.t. \quad & c(x) = 0, g(x) \leq 0 \\ & y, z \geq 0, y_i z_i = \epsilon, i = 1, \ldots, n_c \end{aligned} \tag{6.16}$$

$$\begin{aligned} PF(\rho): \quad \min \quad & f(x) + \rho y^T z \\ s.t. \quad & c(x) = 0,\ g(x) \leq 0,\ y, z \geq 0 \end{aligned} \tag{6.17}$$

Besides the regularization and penalty formulas, nonlinear complementarity problem functions and smoothing functions have also been used to solve MPCCs [87, 88]. An alternative way to form the complementarity constraint is

$$y - max(0, y - z) = 0,\ y > 0,\ z > 0$$

The max operator can be smoothed by several nonlinear functions. One widely studied NCP function is the Fischer-Burmeister function, and the problem is reformulated as follows,

$$\begin{aligned} NCP(\epsilon): \quad \min \quad & f(x) \\ s.t. \quad & c(x) = 0, g(x) \leq 0, y, z \geq 0 \\ & y + z - \sqrt{y^2 + z^2 + \epsilon} = 0 \end{aligned} \tag{6.18}$$

The practical performance of popular NLP reformulations was compared by [75]. Comparing both on the MacMPEC test set and several chemical process examples, it was found that the penalty reformulation was the most effective. The penalty reformulation transfers the complementarity condition to the objective function, multiplied by a penalty weight. The nonlinear programming solver KNITRO has an interface for MPCCs using the penalty reformulation [1]. The method will be discussed in more detail in later sections.

IPOPT-C [89] is a specialized modification of the NLP solver IPOPT designed to solve MPCCs. This method is closely based on the inequality reformulation (second equation in (6.5)). The right hand side zero is replaced with an ϵ term that is automatically adjusted with the barrier parameter μ. As the two parameters converge to zero, the iterates converge superlinearly to the solution under the assumptions of strong stationarity, MPCC-LICQ, and upper level strict complementarity.

FilterMPEC [90] is another method adapted from an NLP solver, in this case filterSQP. As an active set method, it has advantages in dealing with degeneracy of the NLP reformulation. FilterMPEC uses the third reformulation from (6.5) with the right hand side relaxed as follows:

$$y^T z \leq \delta(y_k^T z_k)^{1+\kappa}$$

Without this relaxation, Fletcher and Leyffer [91] showed that the QP can be infeasible arbitrarily close to a solution of the MPCC. By using this relaxation, this issue is avoided

and convergence can be shown. FilterMPEC has linear convergence in theory but practical performance is very good.

While FilterMPEC addresses the problem of infeasible QP subproblems with a problem specific relaxation, SNOPT [14] has a built in general method for handling this problem that works well for solving the inequality reformulation (third equation in (6.5)). When encountering an infeasible QP subproblem, SNOPT will enter nonlinear elastic mode. In this mode, new positive variables are added to slack nonlinear constraints and the positive slacks are penalized in the objective function. This allows the SQP algorithm to continue despite the poorly posed MPCC constraint set. Practical performance is good, similar to FilterMPEC in speed and robustness [90]. Convergence properties are analysed by [86].

In general, replacing the structural complementarity constraint by a set of equations could trap the algorithm into spurious stationary points (A- M- C-stationary points), which are not necessarily stationary points of the original MPCC. SLPEC Alogrithm [84] solves an LPEC to predict the optimal active set and solves an equality-constrained QP to accelerate local convergence. The global convergence proof shows that this algorithm avoids the spurious stationary and converges to B-stationary points. Alternatively, in [92], another numerically stable approach using a sequence of perturbed problems is considered. These methods can successfully deal the spurious stationarity generated by the KKT-approach for bilevel optimization problems and converges to B-stationary points.

6.3 Auto-adjusting penalty methods

We mentioned above that the penalty reformulation method has the most success of the MPCC to NLP reformulations. The success of SNOPT elastic mode is similarly related. The penalty approach to MPCCs works by solving a sequence of penalty subproblems with the

form (6.17).

The penalty parameter ρ can be adjusted to eventually find a solution of the MPCC. The following two theorems form the basis of the penalty method.

Theorem 6. *[83] If x^* is a strongly stationary point for the MPCC (6.1), then x^* is a stationary point of NLP (6.17) for all ρ sufficiently large.*

Theorem 7. *[86] If x^* is a solution to NLP (6.17) for some $\rho > 0$, and x^* is feasible for MPCC (6.1), then x^* is a strongly stationary solution to (6.1).*

Together, these two theorems tell us that we only need to find a sufficiently large value of ρ, then solving (6.17) is equivalent to solving the MPCC, as long as we find a solution where the penalty term in the objective equals zero (i.e. the solution is feasible for the MPCC). However, as ρ gets large, the NLP (6.17) becomes more poorly scaled, so we prefer to start with a moderate value and increase it until finding a solution.

The penalty approach can be very efficient for MPCCs. With a fixed penalty parameter, there is no guarantee that at the optimal solution of (6.17), the penalty function $y^T z$ is below a desired tolerance. In other words, the solution of (6.17) is not necessarily the solution of (6.1). To avoid this problem, in practice, we give an initial guess of ρ, and converge (6.17). If $y^T z$ is too large, we increase the penalty parameter ρ and try again. Once ρ is sufficiently high, we will converge to a strongly stationary solution of (6.1) as long as MPCC-LICQ holds.

Applying IPOPT to (6.17), the subproblem(ρ, μ) is shown as follows,

$$\begin{aligned} min \quad & f(x) + \rho y^T z - \mu B \\ s.t. \quad & c(x) = 0 \\ & g(x) + s = 0 \end{aligned} \tag{6.19}$$

where $B = \Sigma \ln(y_i) + \Sigma \ln(z_i) + \Sigma \ln(s_i)$.

Using the interior point strategy, problem (6.19) is solved repeatedly as μ is decreased until (6.17) converges. In the penalty reformulation for MPCCs, ρ is increased until the solution of (6.17) is feasible for the original problem. In the following section, we investigate two methods to automatically adjust these two parameters

6.3.1 $\rho(\mu)$ algorithm

The typical way to deal with these two parameters ρ and μ is to solve (6.17) with an NLP solver such as IPOPT, then check the value of $y^T z$. If the value is not within the tolerance, we increase the penalty parameter ρ try again. Note that when we increase the penalty parameter ρ, we start the new NLP problem from the current point instead of the initial point of the MPCC. We call this the $\rho(\mu)$ approach because first we converge on μ, then adjust ρ. The detailed algorithm is shown as follows,

Algorithm $\rho(\mu)$:

1. Initialization: $\rho = \rho_0$, $\mu = \mu_0$
2. Solve subproblem(ρ, μ) Equation (6.19)
3. if PF(ρ) is solved, go to next step, otherwise $\mu := c_1\mu$, go to Step 2.
4. if $y^T z \leq \epsilon_{mpcc}$, terminate, otherwise $\rho := c_2\rho$, go to Step 2.

6.3.2 $\mu(\rho)$ algorithm

Algorithm $\rho(\mu)$ required convergence of problem (6.17) before increasing the penalty parameter ρ. However, sometimes it is clear that ρ is not large enough very early in the NLP solving process. In this case, it may not be worthwhile to fully converge the NLP. Rather we can increase ρ before converging μ. This is called the $\mu(\rho)$ approach, first proposed in

[1].

Algorithm $\mu(\rho)$:

1. Initialization: $\rho = \rho_0$, $\mu = \mu_0$
2. Solve subproblem(ρ, μ) Equation (6.19)
3. if $y^T z \geq \mu^{c_3}$, $\rho := c_2\rho$, otherwise $\mu := c_1\mu$.
4. if PF(ρ) is solved and $y^T z \leq \epsilon_{mpcc}$, terminate. Otherwise go to Step 2.

In both algorithms, $\epsilon_{mpcc} > 0$ is the tolerance of the complementarity error. In practice we can use the same value as the overall tolerance of IPOPT. $0 < c_1 < 1$ is the ratio to decrease barrier parameter μ, while $c_2 > 1$ is the ratio to increase the penalty parameter ρ and $0 < c_3 < 1$ is a tuning parameter. In [1], these values were $c_2 = 10$, $c_3 = 0.4$ for experimental results (c_1 is related to IPOPT settings). However, the best values of the parameters are problem specific. In general, we set c_3 so that $\epsilon_{mpcc} = \mu_{final}^{c_3}$ where μ_{final} is an estimate of the final barrier parameter value required to converge the particular problem within tolerance.

In the above two algorithms, every time we change μ or ρ, the filter used for globalization in IPOPT has to be reset. This is a disadvantage of the penalty method because IPOPT may revisit previous points after updating the penalty.

6.4 Constraint elimination methods

Consider the equality reformulation for the constraint $0 \leq y_i \perp z_i \geq 0$

$$y_i z_i = 0 \tag{6.20a}$$

$$y_i \geq 0 \tag{6.20b}$$

$$z_i \geq 0 \tag{6.20c}$$

There are three cases that this constraint is feasible. Case 1: $y_i = 0$, $z_i > 0$. Case 2: $y_i > 0$, $z_i = 0$. Case 3: $y_i = 0$, $z_i = 0$. Since Case 1 and Case 2 are symmetric, we will only discuss Case 1 and Case 3.

For case 1, both (6.20a) and (6.20b) are active because $y_i = 0$ at this point. The Jacobian matrix of active constraints of the equality reformulation for the pair of complementarity i is shown as follows,

$$\begin{bmatrix} z_i & 0 \\ -1 & 0 \end{bmatrix}$$

where $z_i > 0$, note that the matrix is singular. Because the Newton step from IPOPT satisfies the linearized equality constraints. The constraint solves $y_i = 0$, $y_i dz_i + z_i dy_i = 0$ assures the search direction $dy_i = 0$, so $y_i \geq 0$ is a redundant active constraint that will only contribute degeneracy. We could simply remove the $y_i \geq 0$ for the purposes of calculating the step at this iteration, since the search direction will always satisfy this constraint.

For case 3, $y_i = 0$, $z_i = 0$. The Jacobian matrix $y_i z_i = 0$ is a zero row in the KKT matrix. This doesn't provide any information and the degeneracy makes the KKT system hard to solve. As mentioned in Chapter 4, we can use Big-M structured regularization to eliminate the dependent equality.

In Chapter 4, we discussed how to delete dependent equality constraints in IPOPT. In this chapter, we will discuss how to remove inequality constraints, then propose an al-

gorithm to solve MPCCs which automatically removes the dependent constraints locally. The testing results for this new algorithm are shown in Section 6.6.2.

6.4.1 Removing inequality constraints

In IPOPT, all the inequality constraints are modified with slack variables to become equality constraints and the inequality is transformed into bounds on the slack variables. Therefore, numerically eliminating an inequality constraint is equivalent to removing the variable bound. As we mentioned in Chapter 3, the linear system of the KKT conditions is given as follows,

$$\begin{bmatrix} W_k & A_k & -I \\ (A_k)^T & 0 & 0 \\ V_k & 0 & S_k \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \\ d_k^v \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k\lambda_k - v_k \\ c(x_k) \\ S_k V_k e - \mu_j e \end{pmatrix} \tag{6.21}$$

The presentation in Chapter 3 assumes that all variables have a zero lower bound. Here, we assume arbitrary lower bounds x_L and define the matrix $S_k := diag(x_k - x_L)$. This will make it easier to introduce the concept of eliminating a bound.

To remove one lower bound of x, it is equivalent to set the lower bound $x_L^{(i)}$ to $-\infty$. So in (6.21) we set $s_k^{(i)}$ to $+\infty$. Borrowing the concept from Big-M structured regularization, if we would like to eliminate the lower bound of the i^{th} variable, a large finite number M is used in the implementation instead of $s_L^{(i)}$. When we pivot on the modified S_L, the results are shown as follows,

$$\begin{bmatrix} W_k + \Sigma_k & A_k \\ (A_k)^T & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla\varphi_{\mu_j}(x_k) + A_k\lambda_k \\ c(x_k) \end{pmatrix} \tag{6.22}$$

Both $\Sigma_k = S_k^{-1}V_k$ and $\nabla\varphi_{\mu_j}(x_k)$ are changed due to the modification. $\Sigma_k^{(i)} \to 0$ since $(S_k^{(i)})^{-1}$ goes to zero. Similarly, the corresponding $\mu_j^{(i)}$ is eliminated in $\nabla\varphi_{\mu_j}(x_k)$ as well.

Here, we can calculate the step without a bound. However, the bound push and filter acceptance criteria don't change, which assures that the step is also valid for the original problem.

6.4.2 Algorithm

The overall algorithm for solving the MPCC with IPOPT using constraint elimination is stated in Algorithm III.

Algorithm III

Given: Starting point (x_0, λ_0, v_0) with $x_0, v_0 > 0$; initial value for the barrier parameter $\mu_0 > 0$ and $\delta_w^{last} \leftarrow 0$; Constants $\epsilon_{tol}, \kappa_\theta, \kappa_\epsilon > 0$, $s_\theta > 1$, $s_\varphi \geq 1$, $\gamma_\alpha \in (0, 1]$, $\gamma_\varphi, \gamma_\theta \in (0, 1)$, $\eta_\varphi \in (0, \frac{1}{2})$ and $0 < \bar{\delta}_x^{min} < \bar{\delta}_x^{max}$, $\bar{\delta}_c > 0$, $0 < \kappa_x^- < 1 < \kappa_x^+ < \bar{\kappa}_x^+$, $\kappa_c \geq 0$ $\epsilon_{mpcc} > 0$.

1. *Initialize.* Initialize the filter $\mathcal{F}_0 := \{(\theta, f) \in \mathbb{R}^2 : \theta \geq \theta_{\max}\}$ and the iteration counter $k \leftarrow 0$.

2. *Check convergence for the overall problem.* If $E_0(x_k, \lambda_k, v_k) \leq \epsilon_{tol}$ (with the error estimate E_0 defined in (3.4)), then STOP [CONVERGED].

3. *Check convergence for the barrier problem.* If $E_{\mu_j}(x_k, \lambda_k, v_k) \leq \kappa_\epsilon \mu_j$, then

   3.1. Update $\mu_{j+1} = \max\left\{\epsilon_{tol}/10, \min\left\{\kappa_\mu \mu_j, \mu_j^{\theta_\mu}\right\}\right\}$, and set $j \leftarrow j + 1$

   3.2. Re-initialize the filter $\mathcal{F}_0 := \{(\theta, f) \in \mathbb{R}^2 : \theta \geq \theta_{\max}\}$

   3.3. If $k = 0$ repeat step 3, otherwise continue at step 4.

4. *Compute search direction with regularization.*

   4.1. if $y_i < \epsilon_{mpcc}$ and $z_i < \epsilon_{mpcc}$, apply (4.4) to the equality constraint $y_i z_i = 0$. Go to Step 4.5

4.2. if $y_i < \epsilon_{mpcc}$, apply method in Section 6.4.1 to eliminate the bound $y_i > 0$. Go to Step 4.4

4.3. if $z_i < \epsilon_{mpcc}$, apply method in Section 6.4.1 to eliminate the bound $z_i > 0$. Go to Step 4.4

4.4. Factorize the KKT matrix in (3.8) with $\delta_x = \delta_c = 0$. If the inertia is (n, m, 0), then calculate d_k and go to step 5. Otherwise, continue with step 4.5.

4.5. If the matrix has zero eigenvalues, set $\delta_c = \bar{\delta}_c$ Otherwise, set $\delta_c = 0$

4.6. If $\delta_x^{last} = 0$, set $\delta_x = \bar{\delta}_x^0$, otherwise set $\delta_x = \max\left(\bar{\delta_x}^{min}, \kappa_x^- \delta_x^{last}\right)$

4.7. Attempt to factorize the KKT matrix in (3.8) with δ_x and δ_c. If inertia is correct then calculate d_k and go to step 5. Otherwise continue with step 4.8

4.8. If $\delta_x^{last} = 0$, set $\delta_x = \bar{\kappa}_x^+ \delta_x$, otherwise set $\delta_x = \kappa_x^+ \delta_x$.

4.9. If $\delta_x > \bar{\delta}_x^{max}$, abort the current step and directly go to step 9. Otherwise, go to step 4.7.

5. *Backtracking line search.*

5.1. *Initialize line search.* Calculate α_k^{max}, set $\alpha_{k,0} = min(\alpha_k^{max}, 1)$ and $l \leftarrow 0$.

5.2. *Compute new trial point.* If the trial step size becomes too small, i.e. $\alpha_{k,l} < \alpha_k^{\min}$ with $\alpha_k^{\min}$ defined by (3.10), go to the feasibility restoration phase in step 9. Otherwise, compute the new trial point $x_k(\alpha_{k,l}) = x_k + \alpha_{k,l} d_k$.

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

5.4. *Check sufficient decrease with respect to current iterate.*

5.4.1. *Case I: $\alpha_{k,l}$ is an f-step-size (i.e. SC holds):* If the AC for the objective function holds, accept the trial step and go to step 6. Otherwise, go to step 5.5.

5.4.2. *Case II: $\alpha_{k,l}$ is not an f-step-size (i.e. SC is not satisfied):* If SDC holds, accept the trial step and go to step 6. Otherwise, go to step 5.5.

5.5. *Choose new trial step size.* Set $\alpha_{k,l+1} = \frac{1}{2}\alpha_{k,l}$, $l \leftarrow l + 1$, and go back to step 5.2.

6. *Accept trial point.* Set $\alpha_k := \alpha_{k,l}$ and $x_{k+1} := x_k(\alpha_k)$.

7. *Augment filter if necessary.* If k is not an f-type iteration, augment the filter using (3.9); otherwise leave the filter unchanged, i.e. set $\mathcal{F}_{k+1} := \mathcal{F}_k$.

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

9. *Feasibility restoration phase.* Compute a new iterate x_{k+1} by decreasing the primal infeasibility, so that x_{k+1} satisfies SDC and is acceptable to the filter. Augment the filter using (3.9) (for x_k) and continue with the regular iteration in step 8.

6.5 Implementation

The implementation of auto-adjusting penalty methods and constraint elimination methods is based on the AMPL complementarity constraints interface. In AMPL, the complementary constraints can be defined by the keyword *complements*. Then the information is provided to solvers as pairs of variable and constraints that are perpendicular to each other. The formula is shown as follows,

$$x_l \leq x \leq x_u \perp g_l \leq g(x) \leq g_u \tag{6.23}$$

Note that AMPL guarantees that there are exactly two finite bounds for each complementary constraint. However, there is no guarantee that the bounds are found on opposite sides of the *complements* operator. In our implementation, we only focus on the problems with exactly one finite bound on each side of the complementary constraint. The case

where both bounds are on one side of the complementarity can be reformulated to standard form (6.1) but this reformulation is not supported at this time.

To simplify the notation, assume both finite bounds are lower bounds. Now the complementarity constraints are as follows,

$$0 \leq x - x_l \perp g(x) - g_l \geq 0 \tag{6.24}$$

In IPOPT, all inequality constraints are modified by slack variables and the bounds of the inequalities are transferred to the bounds of the slack variables. Then, (6.24) becomes

$$\begin{aligned} &0 \leq x - x_l \perp s - g_l \geq 0 \\ &g(x) - s = 0 \end{aligned} \tag{6.25}$$

which is equivalent to the formulation discussed in Section (6.1).

Both auto-adjusting penalty methods are implemented in IPOPT 3.12.3 with a new parameter ρ_0. The constraint elimination methods are implemented in MIPOPT with a dependent tolerance $\bar{\epsilon}$. All results are obtained on an Intel Core i7-3770 CPU @ 3.40GHz8 with 7.8 GiB memory. IPOPT is compiled with GNU Fortran 4.8.2 and GCC 4.8.2 with the suggested BLAS library by IPOPT. MA57 3.8.0 is applied to both IPOPT and MIPOPT as the linear solver and all options are set to the default values except tol is relaxed to 10^{-6} for all methods.

6.6 Results

MacMPEC [93] is a MPCC benchmark problem library maintained by Sven Leyffer. We applied both the auto-adjusting penalty methods and constraint elimination method to this library to compare performance.

6.6.1 Auto-adjusting penalty methods

We selected 123 problems from MacMPEC and applied both $\rho(\mu)$ and $\mu(\rho)$ methods. First, we vary the parameter ρ_0 (the initial value of ρ) and plot the performance profile. The performance is measured by iteration count. The optimal solution of each MacMPEC problem is well known and posted on the website. For consistent comparison, we only consider a problem as solved when it converged to the reference optimal solution.

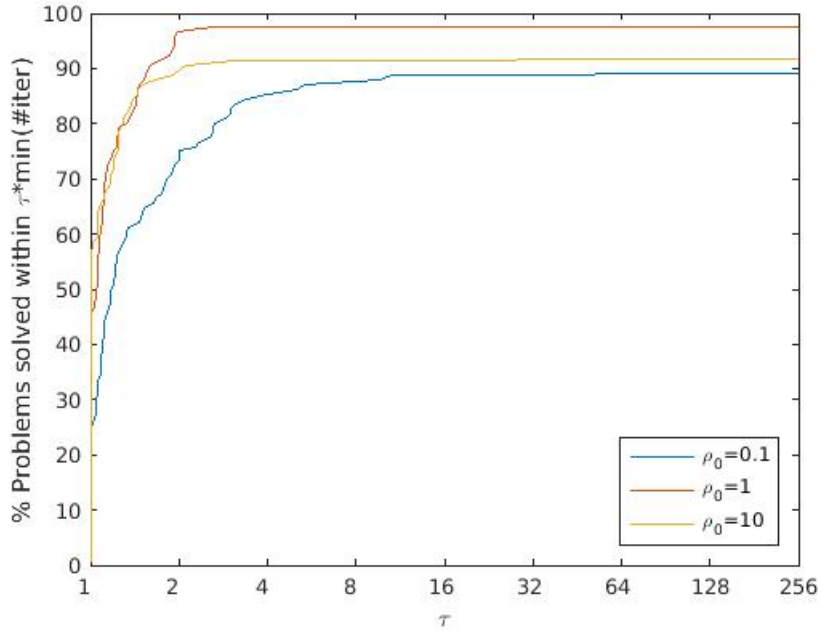

Figure 6.1: performance profile $\rho(\mu)$

The parameter ρ_0 is very important for both algorithms. If it is initialized with a large value, it will put too much weight on satisfying the complementarity constraint, and convergence is slow. However, if ρ_0 starts at a very small value, the initial iterates wander far away from the feasible set before converging.

According to the performance profile, for both $\rho(\mu)$ and $\mu(\rho)$ methods, $\rho_0 = 1$ has the

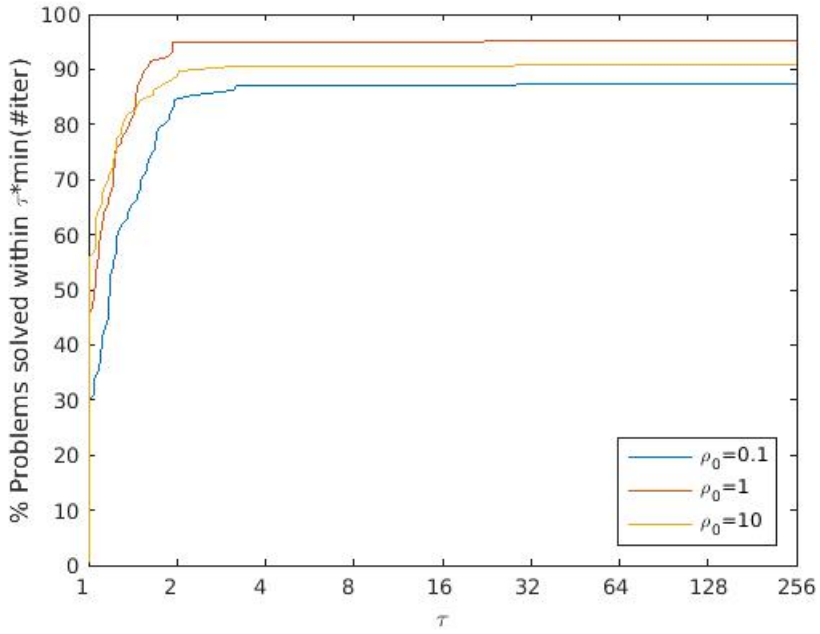

Figure 6.2: performance profile $\mu(\rho)$

best performance, which can be explained by the good scaling of the MacMpec library.

In Fig. 6.3, we compare 5 different methods. These include both $\rho(\mu)$ and $\mu(\rho)$ methods with $\rho_0 = 1$, the equality reformulation $RegEq(0)$ (6.16) with and without structured regularization, and the inequality relaxation $Reg(10^{-4})$ (6.14). $RegEq(0)$ is the worst method in both speed and stability because there is no interior area for IPOPT to explore. Regularization helps with the degeneracy but restoration is often called at nearly feasible points. However, structured regularization as a special case of our constraint elimination method dramatically improves the performance even if it only helps the weakly active case. So the result here provides good motivation for the constraint elimination method.

Among these methods, the $\rho(\mu)$ method is the fastest and the most stable. Also the auto-adjusting penalty methods are much faster than the other methods, with 60% of problems solved with the fewest iterations. For relatively simple problems, penalty approaches gives

IPOPT a lot of freedom to explore and in general will converge faster than $Reg(\epsilon)$. Meanwhile, since the algorithm has to empty the filter either μ or ρ changes, $\rho(\mu)$ method keeps to one ρ value, which persevered the filter information longer than $\mu(\rho)$. This may help explain the faster convergence.

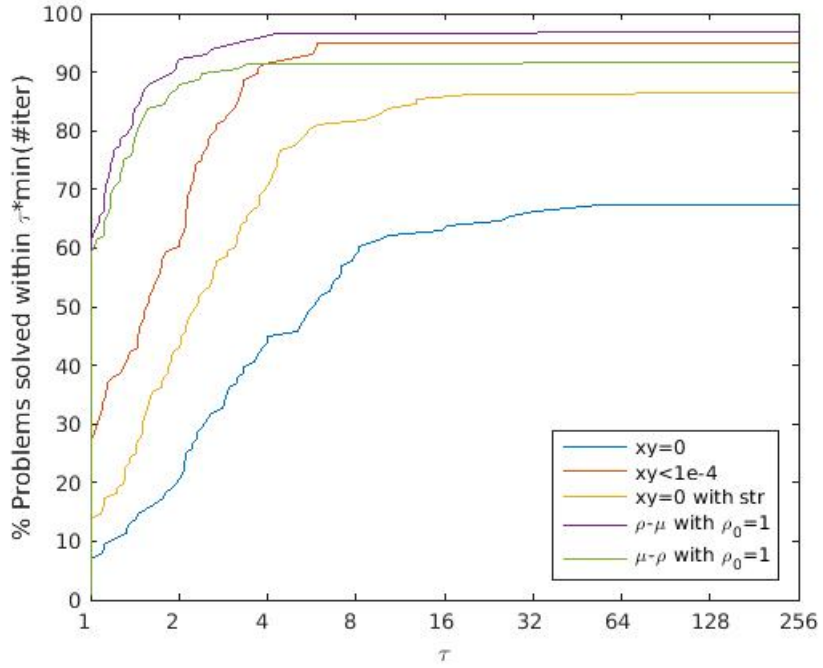

Figure 6.3: performance profile comparison

6.6.2 Constraint elimination methods

In this section, we selected several problems from MacMpec and test it with constraint elimination method. The results are shown in Table 6.1. The entries in the table are the number of iterations required to solve a particular problem with the specified method. 'RF' means restoration failed.

In general, the proposed method converge faster than $RegEq(0)$. For *outrata32* and *out-*

rata33, $RegEq(0)$ calls the restoration phase at an almost feasible point and restoration phase fails immediately. In our new algorithm, the degeneracy is removed by constraint elimination, allowing the algorithm to converge to the optimal solution easily.

The proposed algorithm is very sensitive to the parameter ϵ_{mpcc}, which is used to judge whether a variable bound is active or not. In our experiments, we set ϵ_{mpcc} to 10^{-4} for all problems. However, the appropriate value should be different for each case, or even each iteration. We observe that when ϵ_{mpcc} is too small, restoration phase is called before the algorithm removes the degenerate constraints. In this case, IPOPT will terminate with the error that calling restoration at an almost feasible point. However, when ϵ_{mpcc} is too large, the constraint is removed too early, before it is actually active. This can cause the Newton step to go outside of the bounds. The linesearch cuts back the step, but because the search direction may not be valid, progress may be stuck at that point. These problems dramatically reduce the performance of the constraint elimination methods.

In the future, we would like to further investigate the effect of ϵ_{mpcc} and try to find a systematic way to set it. According to our experience, it may be effective to set ϵ_{mpcc} based on the barrier parameter μ. Since the barrier parameter determines how close iterates can get to bounds, it is related to what we could reasonably consider as an active bound. Therefore as μ changes, we can change the tolerance accordingly.

6.7 Applications

6.7.1 Differential inclusion

Complementarity constraints are useful in many dynamic optimization problems. Because complementarity is required to hold at every point in time, the number of complementarities n_c after discretization can be very large. By increasing the number of discretization

Table 6.1: MacMPEC results (# iter) for constraint elimination method

Test Case	Proposed Algorithm	$RegEq(0)$	Test Case	Proposed Algorithm	$RegEq(0)$
kth1	19	24	exp913	32	30
kth2	10	15	exp915	16	23
kth3	6	10	exp917	33	28
scholtes1	10	14	sl1	25	31
scholtes2	18	23	outrata31	41	86
scholtes 3	8	15	outrata32	40	RF
scholtes 4	22	22	outrata33	33	RF
exp911	23	27	outrata34	45	59

points, n_c can be arbitrarily increased. This test problem uses a dynamic system represented by the differential inclusion

$$\dot{x} \in \text{sgn}(x) + 2$$

with initial condition $x(0) = -2$. This differential inclusion is easily modified into complementarity constraints. Applying the reformulation and adding an arbitrary objective

function, we form the optimal control problem:

$$\begin{aligned} \min \quad & (x_{end} - 5/3)^2 + \int_0^{t_{end}} x^2 dt \\ s.t. \quad & \dot{x} = u + 2 \\ & x(0) = -2 \\ & x = s^+ - s^- \\ & 0 \leq 1 - u \perp s^+ \geq 0 \\ & 0 \leq u + 1 \perp s^- \geq 0 \end{aligned} \tag{6.26}$$

This problem is discretized using the implicit Euler method, as follows:

$$\begin{aligned} \min \quad & (x_{end} - 5/3)^2 + \sum_{i=1}^{N} x_i^2 \\ s.t. \quad & \dot{x}_i = u_i + 2 \quad i = 1, \ldots, N \\ & x_i = x_{i-1} + h\dot{x}_i \quad i = 1, \ldots, N \\ & x_0 = -2 \\ & x_i = s_i^+ - s_i^- \quad i = 1, \ldots, N \\ & 0 \leq 1 - u_i \perp s_i^+ \geq 0 \quad i = 1, \ldots, N \\ & 0 \leq u_i + 1 \perp s_i^- \geq 0 \quad i = 1, \ldots, N \end{aligned} \tag{6.27}$$

We solved this model with

$$N = [10, 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000]$$

When $N = 8000$, the problem has 95940 variables, 47964 equality constraints and 31986 inequality constraints along with 15988 complementary constraints.

With $N = 10$, this problem is solved using the $\rho(\mu)$ and $\mu(\rho)$ methods and compared to $Reg(10^{-5})$, $RegEq(0)$ with and without structured regularization. The performance profile for these 10 problems are shown in Fig 6.4.

Overall, this performance profile is very similar to Fig 6.3. However, $\mu(\rho)$ is slightly better than $\rho(\mu)$ methods. Since these 10 problems are derived from one problem, $\mu(\rho)$

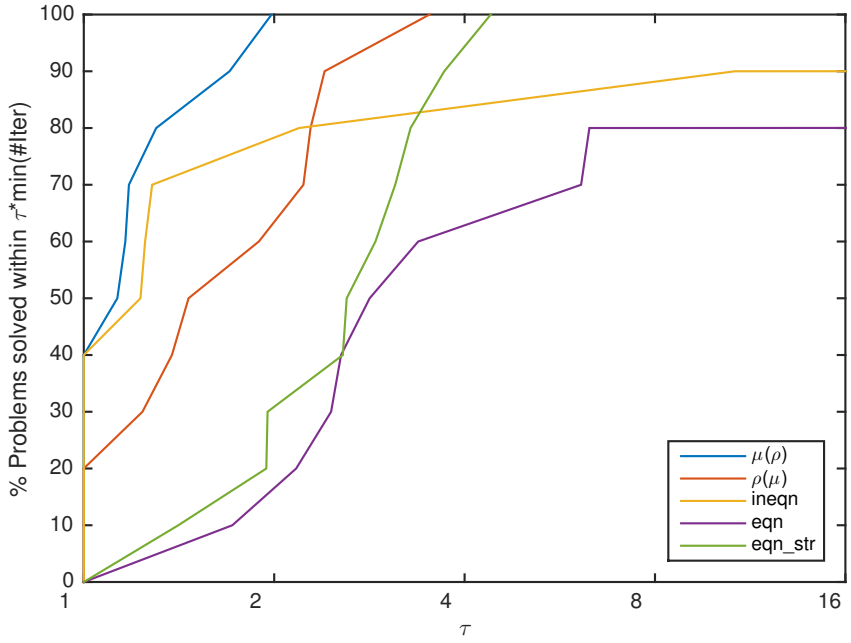

Figure 6.4: performance profile comparison

approach seems more suitable for this problem. $Reg(10^{-5})$ failed in restoration phase for one case, while $RegEq(0)$ failed to converge for two cases. Structured regularization can help equality reformulations, but it is very slow due to the difficulty of recognizing the dependent equality constraints.

6.7.2 Distillation Optimization

A common application of MPCCs in chemical engineering is for phase equilibrium problems. The discontinuous behavior occurs when a phase disappears. The overall behavior of a vapor liquid system is determined by the minimization of Gibbs free energy. The Gibbs free energy minimization problem is written as follows, where the mass balance creates inequality constraints. For fixed temperature T and pressure P , the minimization

determines the amount of chemical species i in vapor phase v_i and liquid phase l_i. The total number of chemical species is NC.

$$\begin{aligned} \min_{l_i,v_i} \quad G(T,P,l_i,v_i) &= \sum_{i=1}^{NC} l_i \bar{G}_i^L + \sum_{i=1}^{NC} v_i \bar{G}_i^V \\ s.t. \quad & \sum_{i=1}^{NC} l_i \geq 0 \quad \sum_{i=1}^{NC} v_i \geq 0 \\ & l_i + v_i = m_i > 0, \quad i = 1, \ldots, NC \end{aligned} \tag{6.28}$$

To include phase equilibrium within a larger model, the optimality conditions of this problem are written using complementarity constraints. After some reformulation using thermodynamic laws, the optimality conditions can be expressed as

$$\begin{aligned} y_j &= \beta K_j x_j \\ \beta &= 1 - s^l + s^v \\ 0 \leq L \perp s^l &\geq 0 \\ 0 \leq V \perp s^v &\geq 0 \end{aligned} \tag{6.29}$$

where $y_j = \frac{v_j}{\sum_{i=1}^{NC} v_i}$ is the vapor fraction, x_j is the liquid fraction, $K_j = \phi_L/\phi_V$ and ϕ_L and ϕ_V are the fugacity coefficients for liquid and vapor respectively. The full derivation can be found in [3].

The formulation (6.29) can be used anywhere that requires vapor liquid equilibrium calculations. In the following section we show how this can be used to optimize a distillation column.

A distillation column can be modeled using the mass balance, equilibrium, summation, and heat balance (MESH) equations. We will consider a modification proposed by Lang and Biegler [94] to optimize the feed tray location and total number of trays. The total number of trays is adjusted by changing the reflux location. Trays above the reflux locations are basically "turned off" by the optimization. To avoid introducing integer variables, the reflux and feed streams are fed to all N_{max} potential trays in the column.

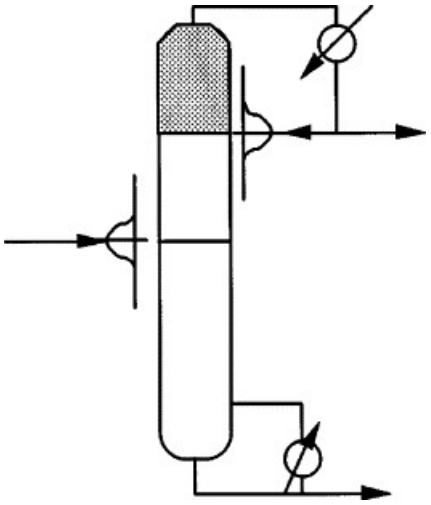

Figure 6.5: Distillation Column

As shown in Figure 6.5, the distribution of these flows is required to be a discretized Gaussian distribution, where the mean is a variable and the standard deviation is fixed. The standard deviation is chosen to be small (0.5) so that most of the flow goes in at the location of the mean. Therefore, the mean of these distributions represent the optimal feed and reflux tray locations.

In the distillation model, each potential tray includes the equilibrium equations (the 'E' in MESH). This means that equations (6.29) are indexed over all potential trays as follows:

$$\begin{aligned} y_{ij} &= \beta_i K_{ij} x_{ij} \\ \beta_i &= 1 - s_i^l + s_i^v \\ 0 \leq L_i &\perp s_i^l \geq 0 \\ 0 \leq V_i &\perp s_i^v \geq 0 \end{aligned} \tag{6.30}$$

where $i = 1 \ldots N_{max}$, j is the component index, L_i and V_i are the liquid and vapor flowrates leaving tray i, and x and y are liquid and vapor mole fractions.

In this section, we will test our methods on a distillation column separating benzene

and toluene. This uses the MPCC model presented above with $N_{max} = 25$, and the feed stream is 100 mol/s of a 70/30 mixture of benzene/toluene. The distillate is required to be 50 mol/s. The model was tested with several different objective functions of the form

$$\min \quad w_t D x_{D,tol} + w_r r + w_n N_t \tag{6.31}$$

where $x_{D,tol}$ is the mole fraction of toluene in the distillate, $D = 50$ is the distillate flowrate, r is the reflux ratio, and N_t is the number of trays. The weights w_t, w_r, and w_n allow us to examine the trade-offs between product purity, utility cost, and capital cost. The model was tested using ideal thermodynamics. The initial point used 21 trays, with the feed tray at tray 7 and reflux ratio of 2.2. Temperature and concentration profiles were linearly interpolated based on top and bottom conditions.

This problem has 48 complementarity constraints, with 403 variables and 350 equality constraints. By changing the objective function weights, three problems were tested.

- Case 1: ($w_t = w_r = w_n = 1$) Equal weights on toluene in distillate.
- Case 2: ($w_t = 1$, $w_r = 0.1$, $w_n = 1$) This case places less weight to utility usage, as represented by the reflux ratio r.
- Case 3: ($w_t = 1$, $w_r = 1$, $w_n = 0.1$) This case places less weight on capital cost, as represented by the number of trays N_t.

We test these 3 cases with $\rho(\mu)$ and $\mu(\rho)$ methods and compared to the NCP reformulation (6.18), inequality relaxation (6.14), equality reformulation (6.16) with and without structured regularization. The results of this comparison are listed in Table 6.2,

In this table, RF stands for "restoration failed" and the number in parentheses shows the iteration that restoration fails. All the other numbers represent the number of iterations for IPOPT to converge to the optimal solution.

From Table 6.2, we see that the NCP reformulation is the hardest one for IPOPT to solve.

Table 6.2: Distillation test results (# iter)

	Case 1	Case 2	Case 3
NCP(10^{-8})	1080	562	410
RegEq(0)	RF(692)	350	RF(687)
RegEq(0) with str	280	382	268
Reg(10^{-5})	115	198	111
Reg(10^{-8})	191	196	118
$\rho(\mu)$	159	173	147
$\mu(\rho)$	205	245	157

The equality reformulation is naturally degenerate. Both restoration failures are due to calling restoration phase at an almost feasible point. Degeneracy occurs as IPOPT approaches a feasible solution. If IPOPT regularization can't correct the inertia, restoration is called and then fails directly. However, if we apply structured regularization instead, all three cases can be solved in a reasonable number of iterations. As discussed before, the inequality relaxation generally works well for IPOPT. Here we compared $Reg(10^{-5})$ and $Reg(10^{-8})$ with our $\rho(\mu)$ and $\mu(\rho)$ methods. Since the problem is relatively easy to converge, the advantage of auto-adjust penalty methods are not obvious compared to the previous example problems.

6.8 Conclusion

In this chapter, we reviewed optimality conditions and CQs and discussed several NLP reformulations for MPCCs. Based on previous success with penalty-based reformulations

with interior point solvers, we implemented two auto-adjustment penalty approaches in IPOPT. These two methods are tested on the MacMPEC library, and we conclude $\rho(\mu)$ is slightly better. We also discussed two applications of MPCCs and compared the auto-reformulation techniques against other popular reformulations. Finally we discussed an extension of structured regularization to MPCCs, called the constraint elimination method. Preliminary results for the constraint elimination show that it can solve problems but numerical difficulties remain. Future work can further explore these challenges for constraint elimination. Also, the penalty approach relies on the assumption of a strong stationary solution. Another potential direction would be to address problems where only B-stationarity holds.

Chapter 7

Conclusions

In this chapter we summarize our contributions and discuss the directions for future work.

7.1 Summary and Contributions

This thesis can be summarized as follows, Chapter 1 defined nonlinear programming problems, and discussed the usefulness in many applications. We also introduced several challenges in nonlinear programming. Chapters 2 and 3 provide a basis for the methods presented in Chapters 4-6. Chapter 2 reviewed first and second order optimality conditions for nonlinear programming. In addition, we talked about the history of NLP solvers and current popular NLP solvers. In Chapter 3, we reviewed the detailed algorithm of IPOPT, including background for interior methods and the combination of line search and filter concepts. The local and global convergence proofs of IPOPT were also discussed. Chapter 4 proposed three structured regularization methods for interior point methods. Dependent equality constraints are common in process system models. With the proposed methods, these constraints can be removed locally from model and effectively without changing the sparsity pattern during the solution process. We implemented these methods in IPOPT and compared them with current versions of IPOPT and KNITRO. The results shows the new regularization methods dramatically increased the performance of IPOPT for dependent equality constraints. Then we applied the new algorithm on two blending problems that are locally degenerate and improvements are observed. The major contributions are

listed as follows,

- Proposed three regularization methods to remove dependent equality constraints locally.
- Proved the convergence of these methods with IPOPT.
- Implemented these methods in the current version of IPOPT
- Compared structured regularization with current version of IPOPT on CUTEr set and applied them to two blending problems.

Chapter 5 explored methods for speeding up IPOPT on dynamic optimization problems through parallelizing the linear solve. The KKT linear system for dynamic optimization problems can be permuted to a block tridiagonal linear system. Two parallel algorithms capable of exploiting this structure were introduced and time complexity of the algorithms was analyzed. Both algorithms were tested with randomly generated linear systems and the results matched well with predictions based on the time complexity analysis. Finally, we studied how the work distributed to each processors and applied the traditional cyclic reduction method to four dynamic optimization applications. IPOPT with the traditional cyclic reduction linear solver can solve all applications easily with a strong improvement on KKT solving time for large dynamic optimization problems.

The major contributions are listed as follows,

- Proposed to use the cyclic reduction algorithm to solve the KKT linear system from dynamic optimization problems.
- Implemented two traditional cyclic reduction method and Yalamov's cyclic reduction method with and without symmetric updates.
- Analysed the time complexity of these two cyclic reduction methods and verified the analysis with performance of cyclic reduction methods on randomly generated block tridiagonal linear systems.

- Implemented a tool to automatically permute dynamic optimization problems to block tridiagonal form.
- Applied traditional cyclic reduction with symmetric updates to four dynamic optimization applications, showing the potential of the parallel method for very large dynamic applications.

Chapter 6 considered the solution of MPCCs using IPOPT. This chapter first reviews the special optimality conditions and constraint qualifications for MPCCs, then introduced popular ways to reformulate MPCCs to NLPs. We implemented two auto-adjustment penalty methods in IPOPT and tested them with a common benchmark MPEC library MacMpec. The test results show that $\rho(\mu)$ method is the fastest and also the most stable method. In addition, we applied structured regularization method to the equality reformulation of MPCCs. The promising results for structured regularization with equality reformulation motivated a constraint elimination method customized for MPCCs. This method eliminates either inequality or equality constraints locally according to the active set. Preliminary results were shown for the convergence of the algorithm.

The major contributions are listed as follows,

- Implemented two auto-adjustment penalty methods in IPOPT that can accept complementarity constraints from the AMPL interface.
- Analyzed the performance of auto-adjustment penalty methods and compared with several popular NLP reformulations using MacMpec benchmark library.
- Proposed a method to remove dependent inequality constraints. Combined with structured regularization methods, this forms a constraint elimination method for MPCCs in IPOPT.
- Preliminary results are shown for the constraint elimination method.

7.2 Recommendations for Future Work

In this section, we provide some recommendations for future work.

7.2.1 Dependent Constraints

In Chapter 4, we proposed structured regularization methods to remove the equality constraints. The results show that the algorithm can successfully recognize the dependent equality constraints and remove them effectively. However, the results improve drastically after tuning the threshold parameter. This is because the dependent threshold should be different for each problem based on the problem scaling. In this thesis, the dependent threshold is tuned manually. Future work includes developing an algorithm which can systemically set the dependent threshold based on the particular features and scaling of the problem.

In chemical engineering models, in addition to dependent equality constraints, the dependent active inequality constraints could keep the algorithm from converging as well. Recall that the definition of LICQ considers the active constraints, which include all equality constraints and also active inequality constraints. In IPOPT, all inequality constraints are converted to variable bounds through the introduction of slacks, however, LICQ can still be violated by the active variable bounds. In Chapter 6, we propose a method to remove bounds of a variable in the context of MPCCs. This method could be extended to general dependent inequality constraints. However, the task of recognizing which bounds are dependent is challenging. A naive approach would be to factorize an extra matrix at each iteration consisting of all active constraints. However, one more factorization per iteration will slow down IPOPT significantly. Therefore, we are still searching for a cheaper approach to identify the dependent inequality constraints to apply our method to elimi-

nate dependent variable bounds in IPOPT.

7.2.2 Cyclic Reduction for Dynamic Optimization

In Chapter 5, we observed that the KKT linear system structure for dynamic optimization problems is block tridiagonal. Therefore, Cyclic Reduction (CR) methods can be applied to solve the KKT linear system in parallel for dynamic optimization problems. We assumed that the inertia of the KKT matrix is always $(n, m, 0)$ when it is passed to the linear solver, which is required for the convergence proof of IPOPT. When using a direct factorization solver, the value of the inertia is obtained as a by-product of factorization. However, if we apply CR, the inertia information is hard to get. Chiang and Zavala have proposed an inertia free method for IPOPT if the inertia information is unavailable [95], which we would like to apply to this algorithm.

Based on the time complexity, Yalamov's Cyclic Reduction is $O(n \log n)$ while Traditional Cyclic Reduction is $O(n)$. However, if we have a very high number of cores, i.e. if the number of cores is greater than the number of blocks on the diagonal, then the bottleneck of the cyclic reduction algorithm is the number of cycles instead of factorizations. Yalamov's Cyclic Reduction has one less synchronization for each cycle and both cyclic reduction methods need $\log n$ cycles, so Yalamov's method should be slightly better than the traditional methods. In this thesis, we were limited by computing resources. However, it would be interesting to see if this prediction holds in practice.

7.2.3 Mathematical Programming with Complementarity Constraints

In Chapter 6, we proposed a constraint elimination algorithm to solve MPCCs. Preliminary results show it can solve problems but numerical difficulties remain. For the future work, first we would like to address the numerical issues. For example, the parameter ϵ_{MPCC}

should be automatically set as a function of the barrier parameter μ. Also the convergence of this method should be considered from a theoretical perspective.

Most of the reformulations in Chapter 6 required the assumption of a strong stationary solution. However, there are many examples showing that it is easy to be trapped in a spurious stationary point, where weaker stationarity conditions apply. For future work, we would like to work on an algorithm which only requires B-stationary solutions, and can avoid other spurious solutions.

Bibliography

[1] S. Leyffer, G. López-Calva, and J. Nocedal, "Interior methods for mathematical programs with complementarity constraints," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 52–77, 2006.

[2] A. Forsgren, P. E. Gill, and M. H. Wright, "Interior methods for nonlinear optimization," *SIAM review*, vol. 44, no. 4, pp. 525–597, 2002.

[3] L. T. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. SIAM, 2010.

[4] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[5] P. E. Gill, W. Murray, and M. H. Wright, "Practical optimization," 1981.

[6] W. Murray, "Sequential quadratic programming methods for large-scale problems," *Computational Optimization and Applications*, vol. 7, no. 1, pp. 127–142, 1997.

[7] L. T. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. SIAM, 2010.

[8] A. V. Fiacco and G. P. McCormick, *Nonlinear programming: sequential unconstrained minimization techniques*. SIAM, 1990.

[9] R. Fletcher and S. Leyffer, "Nonlinear programming without a penalty function," *Mathematical programming*, vol. 91, no. 2, pp. 239–269, 2002.

[10] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Knitro: An integrated package for nonlinear optimization," in *Large-scale nonlinear optimization*, pp. 35–59, Springer, 2006.

[11] K. Schittkowski, "Nlpql: A fortran subroutine solving constrained nonlinear programming problems," *Annals of operations research*, vol. 5, no. 1, pp. 485–500, 1986.

[12] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, "User's guide for npsol (version 4.0): A fortran package for nonlinear programming.," tech. rep., STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB, 1986.

[13] P. Gill, W. Murray, and M. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

[14] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[15] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[16] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An interior point algorithm for large-scale nonlinear programming," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 877–900, 1999.

[17] R. J. Vanderbei, "Loqo: An interior point code for quadratic programming," *Optimization methods and software*, vol. 11, no. 1-4, pp. 451–484, 1999.

[18] A. Drud, "Conopt: A GRG code for large sparse dynamic nonlinear optimization problems," *Mathematical Programming*, vol. 31, no. 2, pp. 153–191, 1985.

[19] B. A. Murtagh and M. A. Saunders, "Minos 5.51 users guide," 1983.

[20] A. R. Conn, G. Gould, and P. L. Toint, *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*, vol. 17. Springer Science & Business Media, 2013.

[21] M. Kočvara and M. Stingl, "Pennon: A code for convex nonlinear and semidefinite programming," *Optimization methods and software*, vol. 18, no. 3, pp. 317–333, 2003.

[22] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Motivation and global convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 1–31, 2005.

[23] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Local convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 32–48, 2005.

[24] I. Quesada and I. Grossmann, "Global optimization of bilinear process networks with multicomponent flows," *Computers & Chemical Engineering*, vol. 19, no. 12, pp. 1219–1242, 1995.

[25] L. T. Biegler, I. E. Grossmann, and A. W. Westerberg, "Systematic methods for chemical process design," 1997.

[26] B. Baumrucker, J. Renfro, and L. T. Biegler, "Mpec problem formulations and solution strategies with chemical engineering applications," *Computers & Chemical Engineering*, vol. 32, no. 12, pp. 2903–2913, 2008.

[27] S. Kameswaran and L. Biegler, "Advantages of nonlinear-programming-based methodologies for inequality path-constrained optimal control problemsa numerical study," *SIAM Journal on Scientific Computing*, vol. 30, no. 2, pp. 957–981, 2008.

[28] R. Fletcher, S. Leyffer, D. Ralph, and S. Scholtes, "Local convergence of sqp methods for mathematical programs with equilibrium constraints," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 259–286, 2006.

[29] M. Y. B. Poku, L. T. Biegler, and J. D. Kelly, "Nonlinear optimization with many degrees of freedom in process engineering," *Industrial & Engineering Chemistry Research*, vol. 43, no. 21, pp. 6803–6812, 2004.

[30] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear," *ACM Trans. Math. Softw.*, vol. 9, pp. 302–325, Sept. 1983.

[31] I. S. Duff, "Ma57—a code for the solution of sparse symmetric definite and indefinite systems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 2, pp. 118–144, 2004.

[32] J. D. Hogg and J. A. Scott, *HSL_MA97: A bit-compatible multifrontal code for sparse symmetric systems*. Science and Technology Facilities Council, 2011.

[33] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," *Computer methods in applied mechanics and engineering*, vol. 184, no. 2, pp. 501–520, 2000.

[34] A. Gupta, "Wsmp: Watson sparse matrix package (part-ii: direct solution of general sparse systems)," tech. rep., Citeseer, 2000.

[35] A. Kuzmin, M. Luisier, and O. Schenk, "Fast methods for computing selected elements of the greens function in massively parallel nanoelectronic device simulations," in *Euro-Par 2013 Parallel Processing* (F. Wolf, B. Mohr, and D. Mey, eds.), vol. 8097 of *Lecture Notes in Computer Science*, pp. 533–544, Springer Berlin Heidelberg, 2013.

[36] I. S. Duff, J. K. Reid, N. Munksgaard, and H. B. Nielsen, "Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite," *IMA Journal of Applied Mathematics*, vol. 23, no. 2, pp. 235–250, 1979.

[37] K. Wang, Z. Shao, Y. Lang, J. Qian, and L. T. Biegler, "Barrier nlp methods with structured regularization for optimization of degenerate optimization problems," *Computers & Chemical Engineering*, vol. 57, pp. 24–29, 2013.

[38] A. W. Dowling and L. T. Biegler, "Degeneracy hunter: An algorithm for determining irreducible sets of degenerate constraints in mathematical programs," in *12th International Symposium on Process Systems Engineering and 25th European Symposium on Computer Aided Process Engineering* (J. K. H. Krist V. Gernaey and R. Gani, eds.), vol. 37 of *Computer Aided Chemical Engineering*, pp. 809 – 814, Elsevier, 2015.

[39] N. I. Gould, D. Orban, and P. L. Toint, "Cuter and sifdec: A constrained and unconstrained testing environment, revisited," *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, no. 4, pp. 373–394, 2003.

[40] F. E. Curtis, J. Nocedal, and A. Wächter, "A matrix-free algorithm for equality constrained optimization problems with rank-deficient jacobians," *SIAM Journal on Optimization*, vol. 20, no. 3, pp. 1224–1249, 2009.

[41] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.

[42] T. Binder, L. Blank, H. G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J. P. Schlöder, and O. von Stryk, *Introduction to Model Based Optimization of Chemical Processes on Moving Horizons*, pp. 295–339. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[43] A. E. Bryson and Y.-C. Ho, *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[44] R. G. L.S. Pontryagin, V. Boltyanskii and E.Mishchenko, *Mathematical theory of optimal processes*. New York, NY: Interscience Publishers Inc., 1962.

[45] V. Vassiliadis, R. Sargent, and C. Pantelides, "Solution of a class of multistage dynamic optimization problems. 1. problems without path constraints," *Industrial & Engineering Chemistry Research*, vol. 33, no. 9, pp. 2111–2122, 1994.

[46] V. Vassiliadis, R. Sargent, and C. Pantelides, "Solution of a class of multistage dynamic optimization problems. 2. problems with path constraints," *Industrial and Engineering Chemistry Research*, vol. 33, pp. 2123–2123, 1994.

[47] U. M. Ascher and L. R. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*. Philadelphia: SIAM, 1998.

[48] A. Armaou and P. D. Christofides, "Dynamic optimization of dissipative pde systems using nonlinear order reduction," *Chemical Engineering Science*, vol. 57, no. 24, pp. 5083–5114, 2002.

[49] M. Yu, D. C. Miller, and L. T. Biegler, "Dynamic reduced order models for simulating bubbling fluidized bed adsorbers," *Industrial & Engineering Chemistry Research*, vol. 54, no. 27, pp. 6959–6974, 2015.

[50] A. Hartwich, K. Stockmann, C. Terboven, S. Feuerriegel, and W. Marquardt, "Parallel sensitivity analysis for efficient large-scale dynamic optimization," *Optimization and Engineering*, vol. 12, no. 4, pp. 489–508, 2011.

[51] D. B. Leineweber, I. Bauer, H. G. Bock, and J. P. Schlöder, "An efficient multiple shooting based reduced sqp strategy for large-scale dynamic process optimization. part 1: theoretical aspects," *Computers & Chemical Engineering*, vol. 27, no. 2, pp. 157–166, 2003.

[52] D. B. Leineweber, A. Schäfer, H. G. Bock, and J. P. Schlöder, "An efficient multiple shooting based reduced sqp strategy for large-scale dynamic process optimization: Part ii: Software aspects and applications," *Computers & chemical engineering*, vol. 27, no. 2, pp. 167–174, 2003.

[53] I. D. Washington and C. L. Swartz, "Design under uncertainty using parallel multiperiod dynamic optimization," *AIChE Journal*, vol. 60, no. 9, pp. 3151–3168, 2014.

[54] V. M. Zavala, C. D. Laird, and L. T. Biegler, "Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems," *Chemical Engineering Science*, vol. 63, no. 19, pp. 4834–4845, 2008.

[55] X. Zhang, R. H. Byrd, and R. B. Schnabel, "Parallel methods for solving nonlinear block bordered systems of equations," *SIAM journal on scientific and statistical computing*, vol. 13, no. 4, pp. 841–859, 1992.

[56] D. Feng and R. B. Schnabel, "Globally convergent parallel algorithms for solving block bordered systems of nonlinear equations," *Optimization Methods and Software*, vol. 2, no. 3-4, pp. 269–295, 1993.

[57] D. P. Word, J. Kang, J. Akesson, and C. D. Laird, "Efficient parallel solution of large-scale nonlinear dynamic optimization problems," *Computational Optimization and Applications*, vol. 59, no. 3, pp. 667–688, 2014.

[58] N. Chiang, C. G. Petra, and V. M. Zavala, "Structured nonconvex optimization of large-scale energy systems using pips-nlp," in *Power Systems Computation Conference (PSCC), 2014*, pp. 1–7, IEEE, 2014.

[59] B. Nicholson, S. Kameswaran, and L. T. Biegler, "Parallel cyclic reduction strategies for dynamic optimization," *submitted for publication*, 2017.

[60] R. W. Hockney, "A fast direct solution of poisson's equation using fourier analysis," *J. ACM*, vol. 12, pp. 95–113, Jan. 1965.

[61] W. Gander and G. H. Golub, "Cyclic reductionhistory and applications," in *Scientific computing (Hong Kong, 1997)*, pp. 73–85, 1997.

[62] P. Yalamov and V. Pavlov, "Stability of the block cyclic reduction," *Linear Algebra and its applications*, vol. 249, no. 1, pp. 341–358, 1996.

[63] G. H. Golub and J. M. Ortega, *Scientific computing: an introduction with parallel computing*. Elsevier, 2014.

[64] T. A. Davis, "Algorithm 832: Umfpack v4. 3—an unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 2, pp. 196–199, 2004.

[65] L. S. Blackford, J. Choi, A. Cleary, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, *et al.*, "Scalapack: a portable linear algebra library for distributed memory computers-design issues and performance," in *Supercomputing, 1996. Proceedings of the 1996 ACM/IEEE Conference on*, pp. 5–5, IEEE, 1996.

[66] R. Fourer, D. M. Gay, and B. W. Kernighan, *Ampl*. Boyd and Fraser, 1993.

[67] C. C. Qu and J. Hahn, "Process monitoring and parameter estimation via unscented kalman filtering," *Journal of Loss Prevention in the Process Industries*, vol. 22, no. 6, pp. 703–709, 2009.

[68] S. Rajaraman, J. Hahn, and M. S. Mannan, "A methodology for fault detection, isolation, and identification for nonlinear processes with parametric uncertainties," *Industrial & Engineering Chemistry Research*, vol. 43, no. 21, pp. 6774–6786, 2004.

[69] K.-U. Klatt and S. Engell, "Gain-scheduling trajectory control of a continuous stirred tank reactor," *Computers & Chemical Engineering*, vol. 22, no. 4, pp. 491–502, 1998.

[70] D. Kunii and O. Levenspiel, *Fluidization engineering*. Stoneham, MA: Butterworth-Heinemann, 2 ed., 1991.

[71] S. Mori and C. Wen, "Estimation of bubble diameter in gaseous fluidized beds," *AIChE Journal*, vol. 21, no. 1, pp. 109–115, 1975.

[72] A. Lee and D. C. Miller, "A one-dimensional (1-d) three-region model for a bubbling fluidized-bed adsorber," *Industrial & Engineering Chemistry Research*, vol. 52, no. 1, pp. 469–484, 2012.

[73] S. Modekurti, D. Bhattacharyya, and S. E. Zitney, "Dynamic modeling and control studies of a two-stage bubbling fluidized bed adsorber-reactor for solid–sorbent co2 capture," *Industrial & Engineering Chemistry Research*, vol. 52, no. 30, pp. 10250–10260, 2013.

[74] M. C. Ferris and J.-S. Pang, "Engineering and economic applications of complementarity problems," *Siam Review*, vol. 39, no. 4, pp. 669–713, 1997.

[75] B. Baumrucker, J. Renfro, and L. Biegler, "Mpec problem formulations and solution strategies with chemical engineering applications," *Computers and Chemical Engineering*, vol. 32, no. 12, pp. 2903–2913, 2008.

[76] T. Hoheisel, C. Kanzow, and A. Schwartz, "Theoretical and numerical comparison of relaxation methods for mathematical programs with complementarity constraints," *Mathematical Programming*, pp. 1–32, 2013.

[77] V. DeMiguel, M. P. Friedlander, F. J. Nogales, and S. Scholtes, "A two-sided relaxation

scheme for mathematical programs with equilibrium constraints," *SIAM Journal on Optimization*, vol. 16, no. 2, pp. 587–609, 2005.

[78] F. Facchinei, H. Jiang, and L. Qi, "A smoothing method for mathematical programs with equilibrium constraints," *Mathematical programming*, vol. 85, no. 1, pp. 107–134, 1999.

[79] O. Stein, "Lifting mathematical programs with complementarity constraints," *Mathematical programming*, vol. 131, no. 1, pp. 71–94, 2012.

[80] A. F. Izmailov, A. Pogosyan, and M. V. Solodov, "Semismooth newton method for the lifted reformulation of mathematical programs with complementarity constraints," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 199–221, 2012.

[81] Z.-Q. Luo, J.-S. Pang, and D. Ralph, *Mathematical programs with equilibrium constraints*. Cambridge University Press, 1996.

[82] X. Hu and D. Ralph, "Convergence of a penalty method for mathematical programming with complementarity constraints," *Journal of Optimization Theory and Applications*, vol. 123, no. 2, pp. 365–390, 2004.

[83] D. Ralph and S. J. Wright, "Some properties of regularization and penalization schemes for mpecs," *Optimization Methods and Software*, vol. 19, no. 5, pp. 527–556, 2004.

[84] S. Leyffer and T. S. Munson, "A globally convergent filter method for mpecs," *Preprint ANL/MCS-P1457-0907, Argonne National Laboratory, Mathematics and Computer Science Division*, 2009.

[85] H. Scheel and S. Scholtes, "Mathematical programs with complementarity con-

straints: Stationarity, optimality, and sensitivity," *Mathematics of Operations Research*, vol. 25, no. 1, pp. 1–22, 2000.

[86] M. Anitescu, P. Tseng, and S. J. Wright, "Elastic-mode algorithms for mathematical programs with equilibrium constraints: global convergence and stationarity properties," *Mathematical programming*, vol. 110, no. 2, pp. 337–371, 2007.

[87] C. Kanzow, "Some noninterior continuation methods for linear complementarity problems," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 851–868, 1996.

[88] O. Stein, J. Oldenburg, and W. Marquardt, "Continuous reformulations of discrete–continuous optimization problems," *Computers & chemical engineering*, vol. 28, no. 10, pp. 1951–1966, 2004.

[89] A. U. Raghunathan and L. T. Biegler, "An interior point method for mathematical programs with complementarity constraints (mpccs)," *SIAM Journal on Optimization*, vol. 15, no. 3, pp. 720–750, 2005.

[90] R. Fletcher and S. Leyffer, "Solving mathematical programs with complementarity constraints as nonlinear programs," *Optimization Methods and Software*, vol. 19, no. 1, pp. 15–40, 2004.

[91] R. Fletcher, S. Leyffer, D. Ralph, and S. Scholtes, "Local convergence of sqp methods for mathematical programs with equilibrium constraints," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 259–286, 2006.

[92] A. G. Mersha and S. Dempe, "Direct search algorithm for bilevel programming problems," *Computational Optimization and Applications*, vol. 49, no. 1, pp. 1–15, 2011.

[93] S. Leyffer, "Macmpec: Ampl collection of mpecs," *Argonne National Laboratory*, 2000.

[94] Y.-D. Lang and L. Biegler, "Distributed stream method for tray optimization," *AIChE Journal*, vol. 48, no. 3, pp. 582–595, 2002.

[95] N.-Y. Chiang and V. M. Zavala, "An inertia-free filter line-search algorithm for large-scale nonlinear programming," *Computational Optimization and Applications*, vol. 64, no. 2, pp. 327–354, 2016.