PARALLEL ALGORITHMS FOR NONLINEAR PROGRAMMING AND APPLICATIONS IN PHARMACEUTICAL MANUFACTURING

Yankai Cao
Purdue University

Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Yankai Cao

Entitled
PARALLEL ALGORITHMS FOR NONLINEAR PROGRAMMING AND APPLICATIONS IN PHARMACEUTICAL MANUFACTURING

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Carl Laird Chair	
Zoltan Nagy	
Gintaras Reklaitis	
Andrew Liu	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Carl Laird

Approved by: John Morgan 12/7/2015
Head of the Departmental Graduate Program Date

PARALLEL ALGORITHMS FOR NONLINEAR PROGRAMMING AND APPLICATIONS IN PHARMACEUTICAL MANUFACTURING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yankai Cao

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2015

Purdue University

West Lafayette, Indiana

To my father Jianjiang Cao, my mother Shuihua Zhu, and my primary school math teacher Yongyuan Diao for inspiring my interest in learning

ACKNOWLEDGMENTS

I am extremely blessed with the opportunities to commit to my Ph.D. study starting at Texas A&M University and completing at Purdue University. Before embarking on the Ph.D., I was a perplexed student with a bachelor's degree in a major I was not interested in and had no idea what I wanted to do and how to start over. Nonetheless, towards the end of graduate school, I am passionate about my research, confident about my future, and become a better person. It is the considerable help and guidance from so many people that makes this transition possible.

I am extremely grateful to have the opportunity to work with my advisor, Dr. Carl Laird. His patience and guidance have helped me to launch my research in the area of control and optimization completely from scratch. If my real life were an optimization problem, although the algorithms he taught me cannot solve this problem because these algorithms all require derivation information which is often not provided in real life (maybe he should teach me genetic algorithms), Dr. Laird has totally redefined my problem formulation. First of all, he has significantly enlarged my feasible region. With the numerous opportunities provided, a lot of things I had seen as infeasible became feasible. Moreover, he is also a perfect role model helping me realize the importance of stepping out of my comfort zone. Most importantly, he has also redefined my utility function. At the start of graduate school, I just wanted to earn a decent job. However, his enthusiasm and dedication to research and teaching, and his various acts of kindness have motivated me to contribute more to the society.

I greatly thank my committee members, Dr. Zoltan Nagy, Dr. Gintaras Reklaitis, and Dr. Andrew Lu Liu for their support and constructive comments. A special thanks goes to Dr. Zoltan Nagy for inspiring me to learn about model predictive control. I would also like to extend my thanks to Dr. Juergen Hahn for his support and being a great role model. He was my co-advisor before he transferred to the Rensselaer Polytechnic Institute and I transferred to Purdue University.

I am grateful to Dr. Victor Zavala and Dr. Naiyuan Chiang for enlightening conversations on stochastic programming during my internships at Argonne national lab. I would also like to thank Gizem Keysan and Shengzhi Shao for their timely advice when I interned at United Airlines. Thanks are also extended to my supervisors and colleagues when I interned at Air Products.

I greatly appreciate all the research group mates for establishing a friendly and constructive research atmosphere. I would appreciate especially those with whom I collaborate closely, Daniel Word, Jia Kang, Arpan Seth, Chen Wang, Gabriel Hackebeil, Jianfeng Liu, Michael Bynum, Jose Santiago Rodriguez, and Todd Zhen.

This work would never be possible without the unconditional love, encouragement and support from my family. My parents taught me the value of patience and diligence. Their strong belief in education motivated my graduate study. My wife Tiantian's companion adds incredible joy and meanings to my life. I really admire her persistence and courage, which inspires me to make tough decisions.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

ABSTRACT

Cao, Yankai PhD, Purdue University, December 2015. Parallel Algorithms for Nonlinear Programming and Applications in Pharmaceutical Manufacturing. Major Professor: Carl D. Laird.

Effective manufacturing of pharmaceuticals presents a number of challenging optimization problems due to complex distributed, time-independent models and the need to handle uncertainty. These challenges are multiplied when real-time solutions are required. The demand for fast solution of nonlinear optimization problems, coupled with the emergence of new concurrent computing architectures, drives the need for parallel algorithms to solve challenging NLP problems. The goal of this work is the development of parallel algorithms for nonlinear programming problems on different computing architectures, and the application of large-scale nonlinear programming on challenging problems in pharmaceutical manufacturing.

The focus of this dissertation is our completed work on an augmented Lagrangian algorithm for parallel solution of general NLP problems on graphics processing units and a clustering-based preconditioning strategy for stochastic programs within an interior-point framework on distributed memory machines.

Our augmented Lagrangian interior-point approach for general NLP problems is iterative at three levels. The first level replaces the original problem by a sequence of bound-constrained optimization problems. Each of these bound-constrained problems is solved using a nonlinear interior-point method. Inside the interior-point method, the barrier subproblems are solved using a variation of Newton's method, where the linear system is solved using a preconditioned conjugate gradient (PCG) method. The primary advantage of this algorithm is that it allows use of the PCG method, which can be implemented efficiently on a GPU in parallel. This algorithm shows an order of magnitude speedup on certain problems.

We also present a clustering-based preconditioning strategy for stochastic programs. The key idea is to perform adaptive clustering of scenarios inside-the-solver based on their influence on the problem. We derive spectral and error properties for the preconditioner and demonstrate that scenario compression rates of up to 94% can be obtained, leading to drastic computational savings. A speed up factor of 42 is obtained with our parallel implementation on an stochastic market-clearing problem for the entire Illinois power grid system.

In addition, we discuss an important application of nonlinear programming in control of pharmaceutical manufacturing processes. First, we focus on the development of real-time feasible multi-objective optimization based NMPC-MHE formulations for batch crystallization processes to control the crystal size and shape distribution. At each sampling instance, based on a nonlinear DAE model, an estimation problem estimates unknown states and parameters and an optimal control problem determines the optimal input profiles. Both DAE-constrained optimization problems are solved by discretizing the system using Radau collocation and optimizing the resulting algebraic nonlinear problem using Ipopt. NMPC-MHE is shown to provide better setpoint tracking than the open-loop optimal control strategy in terms of setpoint change, system noise, and model/plant mismatch. Second, to deal with the parameter uncertainties in the crystallization model, we also develop a real-time feasible robust NMPC formulation. The size of optimization problems arising from the robust NMPC becomes too large to be solved by a serial solver. Therefore, we use a parallel algorithm to ensure real-time feasibility.

1. INTRODUCTION[1]

Fast solution of nonlinear programming (NLP) problems is important in a number of different application areas, including many online or real-time applications. The objective of this dissertation is to develop parallel algorithms to solve large-scale nonlinear programming (NLP) problems. The motivation of developing these algorithms is to make optimal decisions for model-based operations and, in particularly, pharmaceutical manufacturing. This chapter provides an overview of the the current state-of-art of parallel NLP algorithms, and an outline of the thesis.

1.1 Nonlinear Programming and Stochastic Programming

Optimization is an important tool to make decisions across industries. Investment banks use this tool to select portfolios with high expected return while avoiding high risks. Airline companies use this tool to assign planes with different capacity to a number of flights with different demands. Electricity companies use this tool to decide the output of various generators to meet the demand with the lowest cost subjected to transmission constraints. Chemical companies use this tool to decide the control strategy to optimize the product quality. Within the field of optimization, nonlinear

[1]Part of this section is reprinted with permission from “An Augmented Lagrangian Interior-Point Approach for Large-Scale NLP Problems on Graphics Processing Units” by Cao, Y., Seth, A., Laird, C.D., 2015. To Appear, Computers and Chemical Engineering, Copyright 2015 by Elsevier.

optimization is one important area. The goal of nonlinear optimization is to solve the following general nonlinear programming (NLP) problem:

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

where x are the variables, and the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and equality constraints $c : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable. Both f and c can be nonconvex. The vectors x_L and x_U are the lower and upper bounds on x. For this discussion, at any local minimum x^*, we assume that the linear independence constraint qualification (LICQ) and second-order sufficient conditions (SOSC) hold. Problems with general inequality constraints can be transformed to this form with the introduction of slack variables.

To simplify the notation, we consider a problem of the form

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) && && (1.2a)\\ \text{s.t.} \quad & c(x) = 0 && (\lambda) && (1.2b)\\ & x \ge 0. && (\nu) && (1.2c) \end{aligned}$$

Here, $\lambda \in \Re^m$ and $\nu \in \Re^n_+$ are the dual variables for the equality constraints and the bounds. Algorithms that can solve the problem (1.2) can typically solve the general form (1.1) with a few modifications. Furthermore, problem of the form in (1.1) can be transformed to the form in (1.2), so there is no loss of generality in the discussion.

Nonlinear optimization is widely used in chemical engineering for problems ranging from optimal design to optimal operations. The last two decades have witnessed a wide development of nonlinear models based on the first principles. Nonlinear models have the advantages of higher fidelity and larger range of validity. The dynamics of chemical reactions are often described as differential-algebraic equations (DAEs), which can be discretized into a set a large-scale equations. Optimal real-time operations based on these large scale nonlinear models pushes the need for increasingly powerful NLP solvers. Apart from its wide applications in industry, nonlinear optimization is also an essential component in developing many algorithms for mixed integer nonlinear programming (MINLP) problems. A nonlinear solver is often called hundreds of times to solve an MINLP problem. Therefore, an efficient NLP solver can also accelerate the solution time of an MINLP solver significantly.

Despite the high fidelity using the nonlinear models based on the first principles, there are still uncertainties associated with external and internal disturbances. A decision made without consideration of these uncertainties might not only result in low-quality products but also carry the risk of violating some safety constraints. To deal with these uncertainties, we need to solve the two-stage stochastic program of the form

$$\min_{x \in \mathbb{R}^{n_0}} \; f_0(x_0) + E[Q(x_0, p)] \tag{1.3a}$$

$$\text{s.t.} \;\; c_0(x_0) = 0 \tag{1.3b}$$

$$x_0 \geq 0 \tag{1.3c}$$

Here, $x_0 \in \mathbb{R}^{n_0}$ are the first stage variables, p are the parameters with uncertainties following a known distribution on the set $\mathcal{P} \in \mathbb{R}^{n_p}$. The realization of uncertain parameters remains unknown until the second stage. $Q(x_0, p)$ is the optimal value of the second stage problem

$$\min_{x_p} f_p(x_0, x_p) \tag{1.4a}$$

$$\text{s.t.} \;\; c_p(x_0, x_p) = 0 \tag{1.4b}$$

$$x_s \geq 0 \tag{1.4c}$$

Here, x_p are the second stage variables and the form of f_p and c_p may depend on the realization of p.

To solve the problem (1.3) numerically, one method is to assume that p has a finite number of realizations $p_1, ..., p_S$, with probability $\xi_1, ..., \xi_S$. $\mathcal{S} := \{1..S\}$ is the scenario set and S is the number of scenarios. With this assumption,

$$E[Q(x_0, p)] = \sum_{s \in \mathcal{S}} \xi_s \, Q(x_0, p_s). \tag{1.5}$$

Then we can derive the following deterministic equivalent of the two stage stochastic programs and also drop ξ_s from the notation by defining $f_s \leftarrow \xi_s \, f_s$

$$\begin{aligned}
&\min \; f_0(x_0) + \sum_{s \in \mathcal{S}} f_s(x_s, x_0) && && \text{(1.6a)}\\
&\text{s.t.} \;\; c_0(x_0) = 0 && (\lambda_0) && \text{(1.6b)}\\
&\quad c_s(x_0, x_s) = 0 && (\lambda_s), \quad s \in \mathcal{S} && \text{(1.6c)}\\
&\quad x_0 \geq 0 && (\nu_0) && \text{(1.6d)}\\
&\quad x_s \geq 0 && (\nu_s), \quad s \in \mathcal{S}. && \text{(1.6e)}
\end{aligned}$$

Here, x_s is the second stage variable for scenario s, $\lambda_0 \in \Re^{m_0}$ and $\nu_0 \in \Re^{n_0}$ are the dual variables for the first stage equality constraints and the bounds, and $\lambda_s \in \Re^{m_s}$ and $\nu_s \in \Re^{n_s}$ are the dual variables for the second stage equality constraints and the bounds. The total number of variables is $n := n_0 + \sum_{s \in \mathcal{S}} n_s$ and the total number of equality constraints is $m := m_0 + \sum_{s \in \mathcal{S}} m_s$. If we denote $x^T := [x_0^T, x_1^T, ..., x_S^T]$, this problem is a general NLP problem. However, specific solvers can be developed to take advantage of the problem structures.

In many cases, the number of possible realizations of p is infinite. To deal with that situation, often a number of scenarios are generated using Monte Carlo simulation. Although Equation (1.5) is no longer valid by definition, it is often a good approximate when the number of scenarios is sufficiently large. This method is called the Sample Average Approximation (SAA) method. The optimal value from the deterministic equivalent problem (1.6) converges to that of the original problem(1.3) with probability 1 as $S \to \infty$ (Shapiro et al., 2014).

Problem formulation like that in (1.6) can become prohibitively large, especially with large distributed models on large scenario sets. Fortunately, these problems are inherently structured, and several strategies exit for more efficient algorithm that can exploit the structure. We will refer to this class of problems as structured NLP problems.

1.2 Nonlinear Model Predictive Control and Robust Nonlinear Model Predictive Control

One application of NLP is nonlinear model predictive control (NMPC) and one application of structured NLP is robust NMPC. Linear MPC has been a popular advanced control strategy in industry for many years (Qin and Badgwell, 2003). Because of the advances in both computational power and optimization algorithms, nonlinear model predictive control has become more computational feasible, and is advantageous for inherently nonlinear systems to achieve higher product quality and satisfy tighter regulations (Rawlings, 2000; Mayne et al., 2000). The basic idea of NMPC is to solve an optimal control problem at each sampling instance with the updated measured or estimated states. The control values for only the next sampling instance are implemented and the entire process is repeated in the next sampling cycle. For batch processes, since our real interest is in the product quality at the end of the batch, an end-point based shrinking horizon NMPC formulation is frequently used. The full process interval $[t_0,\, t_f]$ can be discretized into N steps. At a sampling instance t_k, the following optimal control problem is solved online:

$$\min_{u(t)} \quad \|y(t_f) - y_{set}\|_{\Pi}^2 \tag{1.7a}$$

$$\text{s.t.} \quad \frac{dz(t)}{dt} = f(z(t), u(t)) \tag{1.7b}$$

$$y(t) = c(z(t), u(t)) \tag{1.7c}$$

$$z(t_k) = \hat{z}(t_k) \tag{1.7d}$$

$$g(z(t), u(t)) \leq 0, t \in [t_k, t_f]\,, \tag{1.7e}$$

where t is time, t_0 and t_f are the start time and end time of the process, $z(t)$ is the vector of state variables, $y(t)$ is the vector of output variables, u represents the manipulated variable temperature, $\hat{z}(t_k)$ is a vector of measured or estimated states at t_k, Π is a weight matrix, and y_{set} are the setpoint values we want to achieve at the end of the batch. Although the whole input profile in the interval $[t_k,\, t_f]$ is computed, only the control action in the interval $[t_k,\, t_{k+1})$ is implemented. At the next sampling instance t_{k+1}, the control horizon shrinks from $[t_k,\, t_f]$ to $[t_{k+1},\, t_f]$, and the optimal control problem is re-evaluated with new measurements and updated state estimates. This DAE-constrained optimization problem can be solved by discretizing the system using Radau collocation on finite elements and optimizing the resulting algebraic nonlinear problem using a general NLP solver.

The quality of the NMPC approach depends on the accuracy of the underlying model. Despite the high fidelity of using the nonlinear models based on the first principles, there are still uncertainties associated with external (e.g. price or demand) and internal (unexplained phenomena) disturbances. One approach to take those uncertainties into consideration in the design of NMPC is to solve the following stochastic program online at each sampling instance t_k

$$\begin{aligned}
\min_{u(t)} \quad & \sum_{s \in \mathcal{S}} \|y_s(t_f) - y_{set}\|_\Pi^2 && \text{(1.8a)} \\
\text{s.t.} \quad & \frac{dz_s(t)}{dt} = h(z_s(t), u(t), p_s) && \text{(1.8b)} \\
& y_s(t) = c(z_s(t), u(t)) && \text{(1.8c)} \\
& z_s(t_k) = \hat{z}(t_k) && \text{(1.8d)} \\
& g(z_s(t), u(t)) \leq 0, && \text{(1.8e)} \\
& t \in [t_k, t_f], \forall s \in \mathcal{S}, && \text{(1.8f)}
\end{aligned}$$

where z_s is a vector of states corresponding to scenario s and parameter value p_s. The control profile u needs to be determined before the true value of p is realized. In the context of stochastic programming, we can view u as the first stage variables and z_s and y_s as the second stage variables. The objective function here minimizes the expected deviation of the product quality at the end of the batch from the desired product qualities. The formulation minimizing the worst case deviation is given in Equation (6.1). This DAE-constrained optimization problem can also be discretized using Radau collocation and the resulting problem is a stochastic programming problem. This is then a highly structured problem that is appropriate for parallel decomposition strategies.

1.3 Overview of Parallel Architectures

Fast solution of nonlinear programming (NLP) problems is important in a number of different application areas, including many online or real-time applications. However, over the past decade, we have seen a fundamental change in computing hardware, and previously observed exponential increases in CPU clock rate have stagnated. While clock rate is not the only determining factor in CPU performance, it is clear that CPU manufacturers have shifted their focus towards multi-core and other parallel computing architectures even for everyday computing needs. The need for increasingly powerful NLP solvers, coupled with the introduction of low-cost parallel computing architectures, heightens the need for the development of NLP algorithms that can utilize these emerging parallel architectures. To design an efficient parallel algorithm, we discuss the advantages and limitations of various parallel architectures.

According to Flynn's taxonomy, there are two typical parallel architectures: multiple-instruction-multiple-data (MIMD) architectures and single-instruction-multiple-data (SIMD) architectures. MIMD architectures can simultaneously execute different instructions in parallel. Two relatively popular classes of MIMD architectures are distributed computing clusters (e.g. beowulf cluster) and multi-core (or shared memory) machines. A distributed computer is typically composed of more than 100 processors connected by a network. For traditional desktop computing needs, the price of distributed computers are generally prohibitive. Furthermore, the communication overhead between processors can lead to efficiency bottlenecks. On the other hand, a multi-core or shared memory architecture provides multiple processing units that share common memory within a single machine. This reduces communication costs, but the possible gains in performance are limited by the number of cores, which is typically small, and potential bottlenecks with multiple processes accessing shared memory.

With single-instruction-multiple-data (SIMD) architectures, each thread can operate on different data, but must execute the same fundamental instruction. Graphics processing units (GPUs), one of the most widely used SIMD architectures available, are emerging as massively parallel systems that offer a large degree of parallelism at relatively low cost. For example, the NVIDIA GPU Tesla K20X, which sold for approximately three thousand US dollars provides 2688 cores capable of 3.95 teraflops for single-precision operation and 1.31 teraflops for double-precision operation. GPUs like the NVIDIA Tesla K20X are not pure SIMD architectures, but rather a hybrid architecture that is composed of a number of independent Streaming Multiprocessors (SMs), each containing several CUDA (Compute Unified Device Architecture) cores. Nevertheless, efficient performance requires that we make use of the parallelism available in the SIMD components and it is reasonable to consider them as such. GPUs offer a much higher level of parallelism than desktop multi-core architectures, are much cheaper than distributed computers, and can be programmed using the rapidly maturing GPU APIs integrated with commonly used languages like C and C++. These advances in GPU technology make it a viable and highly accessible architecture upon which to build numerical algorithms. Over the past several years, GPUs have been used to solve problems in various fields, such as air pollution (Jr. et al., 2010), elasticity simulation (Dick et al., 2011), computational fluid dynamics (Corrigan et al., 2011), neurosurgical simulation (Roman et al., 2010), risk assessment (Zhang et al., 2011), partial differential equations (Elble et al., 2010), and bioinformatics (Vouzis and Sahinidis, 2011).

1.4 Overview of Parallel NLP algorihtms

In order to use parallel architectures to solve NLP problems, numerous algorithms have been proposed. These algorithms can be classified into two categories: one is designed for the general unstructured NLP problems and the other is tailored for particular problem structures such as stochastic programs.

Parallel algorithms for general unstructured NLP problems focus on solving the sparse linear systems to compute the step direction in parallel. For large-scale continuous nonlinear optimization problems, interior-point methods, sequential quadratic programming (SQP) methods and augmented Lagrangian methods are the most successful general purpose algorithms (Nocedal and Wright, 1999). The dominant computational expense in these NLP algorithms is the solution of a large, sparse linear system to generate the step direction at each iteration. A scalable parallel algorithm requires efficient parallel solution of this linear system (along with all other scale dependent operations). There is the possibility for parallel solution of these linear systems using either direct or iterative methods. Amestoy et al. (2000) presents a parallel distributed memory multifrontal approach to solve sparse linear equations.

A speedup of more than 7 is achieved on some test problems with this algorithm. However, the speedup is shown to stagnate with more than 32 processors for most of the test problems. Schenk and Gärtner (2004) shows a parallel sparse unsymmetric LU factorization method integrated into the PARDISO solver for use on shared memory multiprocessor architectures, achieving a speedup of more than 7 on a 8-core machine. Hogg and Scott (2010) develops a symmetric indefinite sparse direct solver within the HSL library for use on multicore machines with OpenMP, achieving a speedup of more than 6 on a 8-core machine. The speedup possible with these approaches is promising, however, the available parallelism is too small for the GPU. Recently, breakthroughs have been made by several researchers, who applied a multifrontal factorization method on a GPU (Krawezik and Poole, 2010; Lucas et al., 2012). The multifrontal method factorizes a sparse matrix by factorizing a tree of dense systems, each of which can be implemented efficiently on a GPU (Galoppo et al., 2005; Agullo et al., 2009; Tomov et al., 2010; Cao et al., 2013). A speedup factor of 2-3 is reported for double precision on a matrix with more than half a million rows/columns. Yeralan et al. (2013) pushes the research further by factorizing those frontal matrices in parallel and achieved up to 10 times speedup on their test problems. However the speedup is not always consistent. For some problems, this algorithm provides limited parallelism, therefore the implementation on a GPU can perform even worse than that on a CPU. In contrast to direct factorization techniques, iterative methods, which require performing simple matrix-vector operations on consistently structured data sets, are highly appropriate for the GPU architecture. The PETSc Library has GPU support for many Krylov Subspace methods with Jacobi, AMG (Algebraic Multigrid), and AINV (Aprroximate Inverse) preconditioners (Kumbhar, 2011). Among iterative methods, the Preconditioned Conjugate Gradient (PCG) method is known to have excellent performance, and several researchers have demonstrated up to 10 times speedup using PCG approaches with different preconditioners on GPUs (Li and Saad, 2013; Helfenstein and Koko, 2011; Buatois et al., 2009).

Developing a parallel solver for general NLP problems also requires an integration of the host NLP algorithm with the parallel linear solver. Parallel linear solvers designed for indefinite matrices can be directly applied to many NLP algorithms. However, to use parallel PCG on a GPU to solve the linear system, the matrix in the linear system needs to be positive definite (P.D.) (Golub and Van Loan, 2012). But the Karush-Kuhn-Tucker (KKT) systems arising from many NLP algorithms are not P.D. For example, in Ipopt (Wächter and Biegler, 2006), the KKT system arising from the interior-point method is a saddle point system and is indefinite even for convex problems.

$$\begin{bmatrix} H_k+\Sigma_k & A_k \\ A_k^T & -D \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \varphi(x_k)+A_k\lambda_k \\ c(x_k) \end{bmatrix}$$

Our work in Cao et al. (2015) proposed an augmented Lagrangian interior-point approach that can use PCG to compute the Newton step in parallel on a GPU. For convex problems, the KKT matrices for subproblems are guaranteed to be P.D. Furthermore, even for non-convex problems, when the variables are near the optimal solution the matrix is also P.D. However, when the variables are far away from the optimal solution, the matrix is not guaranteed to be P.D. Therefore, if the PCG approach detects negative curvature, a diagonal modification to the matrix can be made to ensure that the matrix is positive definite and the step direction is a descent direction.

For structured NLP problems, an efficient parallel algorithm often exploits the structure at problem formulation level (e.g. Bender decomposition, Lagrangian decomposition, Lagrangian relaxation, progressive hedging) or at linear algebra level. Although the parallelization of the first class can be easily implemented, the convergence rate is typically slow for general nonlinear problems. In contrast, the second class of approaches can retain the fast convergence of the original host algorithms. For this class, interior-point methods are popular because the structure of the linear system remains the same at each iteration. The linear systems derived using interior point methods for stochastic programming problems have the block-bordered-diagonal form. These linear systems can be decomposed using the Schur complement method (Zavala et al., 2008). When the number of first stage variables is small, this approach has almost perfect strong scaling. However, when the number of first stage variables is large, forming and solving the dense Schur complement system becomes the bottleneck.

In order to deal with structured NLP problems with large first-stage dimensionality, many approaches have been proposed. Kang et al. (2014) uses a PCG procedure to solve the Schur system with an automatic L-BFGS preconditioner. This approach avoids both forming and factorizing the Schur system explicitly. Lubin et al. (2012) forms the Schur system as a byproduct of a sparse factorization and factorizes the Schur system in parallel. Cao et al. (2015) performs adaptive clustering of scenarios (inside-the-solver) and forms a sparse compressed representation of the large KKT system as a preconditioner. The matrix that needs to be factorized in this approach is much smaller than the full-space KKT system and more sparse than the Schur system.

Besides parallel linear solvers, a scalable parallel algorithm also requires parallel evaluations of the NLP functions and gradients, and parallel implementations of all other linear algebra operations (e.g. vector-vector operations and matrix-vector multiplications). While the latter is easy for many parallel architectures, the former is not. There is, to the best knowledge of the author, no efficient modeling language supporting parallel evaluations of functions and gradients for general NLP problems. Furthermore, the structures of the steaming architecture of the GPU make this very difficult to automate for the general nonlinear case. However, for structured problems, Kang et al. (2014) and Zavala et al. (2008) build one AMPL (Gay and Kernighan, 2002) instance for each scenario and evaluate all instances in parallel. Several packages (e.g. PySP (Watson et al., 2012), StochJuMP(Huchette et al., 2014)) have also been developed to support the parallel evaluation of functions and gradients for structured NLP problems.

1.5 Thesis Outline

Our goal is to develop efficient algorithms for parallel solutions of nonlinear programming problems with applications in pharmaceutical manufacturing. Therefore, this dissertation is organized into two parts.

The first part describes parallel algorithms for NLP problems. Chapter 2 proposes an augmented Lagrangian interior-point approach for general NLP problems that solves in parallel on a Graphics Processing Unit (GPU). Significant speedup is possible on problems with few equality constraints, however, this requires specialized parallel implementations of the model evaluations.

Chapter 3 and Chapter 4 all target on algorithms to solve stochastic programs with distributed memory clusters or multi-core machines. Chapter 3 describes a method to decompose the structured KKT systems using the explicit Schur complement method. When the dimension of first stage variables is small, scalability of this algorithm is almost perfect. However, the cost of forming and factorizing the dense Schur complement matrix increases significantly as the number of first stage variables increases.

In order to solve stochastic programs with a large number of first stage variables, Chapter 4 proposes an algorithm for solving nonlinear stochastic programming problems efficiently through scenario clustering. This approach is unique in that the scenario clustering is applied at the linear solver level, not at the outer NLP level, allowing for scenario clusters to change from iteration to iteration. Furthermore, this clustering approach does not replace the KKT system, but rather is used to build a pre-conditioner. This approach allows one to build a pre-conditioner with fewer clusters, and then solve the full KKT system in parallel using GMRES.

The second part of this dissertation describes the application of nonlinear programming in pharmaceutical manufacturing. Chapter 5 proposes nonlinear model predictive control (NMPC) and nonlinear moving horizon estimation (MHE) formulations for controlling the crystal size and shape distribution in a batch crystallization process. The MHE and NMPC formulations are all DAE-constrained optimization problems that are solved by discretizing the system using Radau collocation on finite elements and optimizing the resulting algebraic nonlinear problem.This model is built in the Modelica modeling language to support solution through the JModelica modeling and optimization framework.

To deal with the parameter uncertainties in the crystallization model, Chapter 6 proposes robust NMPC to minimize the deviation of the product quality from the setpoint in the worst case. The size of these optimization problems becomes too large to be solved by a serial solver, and the algorithm described in Chapter 3 is used to solve the robust NMPC problems.

Finally, Chapter 7 closes this dissertation with conclusions and the directions for future work.

2. AN AUGMENTED LAGRANGIAN INTERIOR-POINT APPROACH FOR LARGE-SCALE NLP PROBLEMS ON GPUS[1]

The demand for fast solution of nonlinear optimization problems, coupled with the emergence of new concurrent computing architectures, drives the need for parallel algorithms to solve challenging nonlinear programming (NLP) problems. In this chapter, we propose an augmented Lagrangian interior-point approach for general NLP problems that solves in parallel on a Graphics processing unit (GPU). The algorithm is iterative at three levels. The first level replaces the original problem by a sequence of bound-constrained optimization problems using an augmented Lagrangian method. Each of these bound-constrained problems is solved using a nonlinear interior-point method. Inside the interior-point method, the barrier sub-problems are solved using a variation of Newton's method, where the linear system is solved using a preconditioned conjugate gradient (PCG) method, which is implemented efficiently on a GPU in parallel. This algorithm shows an order of magnitude speedup on several test problems from the COPS test set.

The chapter is organized as follows. Section 2.1 gives some background information. A description of the overall algorithm is given in Section 2.2. Section 2.3 describes the parallel implementation details of the proposed algorithm, including a discussion of pros and cons for using different matrix storage formats. Section 2.4 presents the numerical performance of the algorithm on some problems selected from the COPS (Dolan et al., 2004) test set including a comparison with the state of the art solver Ipopt. Section 2.5 summarizes this chapter.

[1]Part of this section is reprinted with permission from "An Augmented Lagrangian Interior-Point Approach for Large-Scale NLP Problems on Graphics Processing Units" by Cao, Y., Seth, A., Laird, C.D., 2015. To appear, Computers and Chemical Engineering, Copyright 2015 by Elsevier.

2.1 Preliminaries

Section 1.3 already discusses the advantages and limitations of various parallel architecture. For GPU, because of its specific structure, it is not efficient on factorization of sparse matrix. In contrast to direct factorization techniques, iterative methods, which require performing simple matrix-vector operations on consistently structured data sets, are highly appropriate for the GPU architecture. Among iterative methods, the Preconditioned Conjugate Gradient (PCG) method is known to have excellent performance, and several researchers have demonstrated up to 10 times speedup using PCG approaches with different preconditioners on GPUs (Li and Saad, 2013; Helfenstein and Koko, 2011; Buatois et al., 2009).

However, the desire to use PCG to solve the linear system imposes limitations on the NLP algorithm we can use. PCG requires the matrix in the linear system to be positive definite (P.D.) (Golub and Van Loan, 2012). But Karush-Kuhn-Tucker (KKT) systems arising from many NLP algorithms are not P.D. For example, in IPOPT (Wächter and Biegler, 2006), the KKT system arising from the interior-point method is a saddle point system and is indefinite even for convex problems.

$$\begin{bmatrix} H^k + \Sigma^k & A^k \\ (A^k)^T & -D \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \varphi(x^k) + A^k \lambda^k \\ c(x^k) \end{bmatrix}$$

Hence, PCG cannot be directly applied to this saddle point system. For a saddle point system with $D=0$, Bergamaschi et al. (2004); Lukšan and Vlček (1998); Perugia and Simoncini (2000); Gould et al. (2001) employ a constraint preconditioner that allows the use of the PCG method. Dollar et al. (2006); Dollar (2007) extend the constraint preconditioner for general D. Forsgren et al. (2007) shows that the saddle point system with positive diagonal D can be transformed into the doubly augmented system or condensed system. Provided the original saddle point system has the correct inertia, the augmented system and condensed system can be proven to be positive definite, and the PCG method in conjunction with the constraint preconditioner shows promise. If the PCG method detects negative curvature, it implies the matrix is not positive definite and the intertia condition is not satisfied for the original system. However, a major problem with applying this technique is that the constraint preconditioner requires a sparse matrix factorization and backsolves, for which no currently efficient implementations are available for the general case.

The augmented Lagrangian method moves the equality constraints to the objective function and solve the NLP as a sequence of bound-constrained sub-problems. This method has the benefit that the KKT system is positive definite when the problem is convex. Lancelot (Conn et al., 1988), first released in 1992, is a well-known example of an augmented Lagrangian code for NLP problems. Lancelot uses the gradient projection method to solve the bound-constrained problems. This method first finds the Cauchy point, which is the first local minimizer of the approximation of the objective function along the steepest descent direction (Conn et al., 1988), or a point satisfying sufficient decrease condition (Lin and Moré, 1999). If the bounds are reached before the first minimizer is found, the search direction is bent at the corresponding bounds. Then the gradient projection method fixes the components of the Cauchy point that are at their bounds and performs the subspace minimization with the PCG method. When the bounds are violated in a particular PCG iteration, the PCG method is terminated (Conn et al., 1988). With the gradient projection method, PCG is easily stopped because of the violation of the bounds. In a GPU implementation, changes in the active set (which are performed on the host), might be the dominant computational expense compared to the PCG iterations performed on the GPU. Therefore, gradient projection method is not directly appropriate for parallel implementation on a GPU.

In this chapter, we use an interior-point method to solve the bound-constrained problems by replacing each with a series of unconstrained barrier sub-problems. We can use a variation of Newton's method to solve the unconstrained sub-problem. The primary advantage of this approach is that we can use PCG to compute the Newton's step in parallel on a GPU. For convex problems, the KKT matrix arising from the unconstrained sub-problem is guaranteed to be P.D. Furthermore, even for non-convex problems, when the variables are near the optimal solution the matrix is also P.D. However, when the variables are far away from the optimal solution, the matrix is not guaranteed to be P.D. Therefore, if the PCG approach detects negative curvature, a diagonal modification to the matrix can be made to ensure that the matrix is positive definite and the step direction is a descent direction.

2.2 Algorithm

In this section, we first present the proposed augmented Lagrangian interior-point algorithm to deal with the nonlinear programming problem of the form (1.2). In Section 2.2.5, we discuss the modifications necessary to handle the general form (1.1).

2.2.1 Bound-constrained Augmented Lagrangian Method

The augmented Lagrangian is formed by adding a quadratic penalty term to the Lagrangian function. Neglecting the inequalities, the augmented Lagrangian for eqs. (1.2a) and (1.2b) is

$$\mathcal{L}_A(x, \bar{\lambda}; \mu) = f(x) - \bar{\lambda}^T c(x) + \frac{\mu}{2} c(x)^T c(x), \tag{2.1}$$

where μ is the penalty parameter and $\bar{\lambda}$ is the estimate of the true Lagrange multipliers λ. The augmented Lagrangian approach then computes the solution for a sequence of bound-constrained sub-problems

$$\min_{x \in \mathbb{R}^n} \ \mathcal{L}_A(x, \bar{\lambda}; \mu) \tag{2.2a}$$

$$\text{s.t.} \ \ x \geq 0. \tag{2.2b}$$

This sub-problem is solved approximately for fixed value of $\bar{\lambda}$ and μ. If the violation of the equality constraints has decreased sufficiently, the estimate of the Lagrange multipliers $\bar{\lambda}$ is updated, and the sub-problem tolerance is made tighter. Otherwise, the penalty parameter μ is increased to improve feasibility.

It can be proven (Nocedal and Wright, 2006) that under reasonable assumptions, a good estimate of the optimal solution x^* can be obtained after solving a sequence of sub-problems when μ is large enough or when $\bar{\lambda}$ is a good estimate of the Lagrange multipliers λ. Also, if μ is sufficiently large, then each $\bar{\lambda}$ update will lead to a more accurate estimate of the Lagrange multipliers. It can also be proven that the Hessian of the augmented Lagrangian function is positive definite when x and $\bar{\lambda}$ are sufficiently close to the optimal solution.

The augmented Lagrangian framework forms the outer-loop of the algorithm. At each iteration, we require the solution of the bound-constrained sub-problem.

2.2.2 Interior-Point Method for the Bound-Constrained Sub-Problem

There are several techniques one could use to solve the bound-constrained sub-problem including an active set method, a gradient projection method, or an interior-point method. The use of an interior-point method produces a linear system that can be solved efficiently on a GPU. Hence, we replace the bound-constrained sub-problem by a series of unconstrained barrier sub-problems

$$\min_{x \in \mathbb{R}^n} \quad \phi(x) = \mathcal{L}_A - \mu_{in} \sum_{i=1}^{n} \ln(x^{(i)}), \tag{2.3}$$

where $x^{(i)}$ denotes the ith component of the vector x, and $\mu_{in} > 0$ is the barrier parameter. The first-order optimality conditions of the unconstrained sub-problem are

$$\nabla \phi(x) = \nabla \mathcal{L}_A(x) - \mu_{in} X^{-1} e = 0, \tag{2.4}$$

where X=diag(x), and e is a vector with all elements equal to 1. The primal-dual reformulation can be written by introducing $\nu=\mu_{in}X^{-1}e$,

$$\nabla\mathcal{L}_A(x) - \nu = 0 \tag{2.5a}$$

$$X\nu - \mu_{in}e = 0. \tag{2.5b}$$

When μ_{in} is 0, the above equations together with $x \geq 0, \nu \geq 0$ are the KKT conditions for the bound-constrained sub-problem (2.2). We can use a variation of Newton's method to solve the primal-dual system of Equations (2.5). At each iteration k of the Newton's method, the step direction is calculated by solving the linear system

$$\begin{bmatrix} \nabla^2\mathcal{L}_A & -I \\ V^k & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta\nu^k \end{bmatrix} = -\begin{bmatrix} \nabla\mathcal{L}_A - \nu^k \\ X^k\nu^k - \mu_{in}e \end{bmatrix}. \tag{2.6}$$

Here $\nabla^2\mathcal{L}_A$ denotes the Hessian of the augmented Lagrangian function

$$\nabla^2\mathcal{L}_A = \nabla^2 f(x^k) - \sum(\bar{\lambda_i} - \mu c_i(x^k))\nabla^2 c_i(x^k) + \mu A^k(A^k)^T, \tag{2.7}$$

with A^k:=$\nabla c(x^k)$, while Δx^k and $\Delta\nu^k$ are the search directions in x^k and ν^k respectively. The first two terms of Equation (2.7) can be expressed as $\nabla^2\mathcal{L}(x^k, \bar{\lambda} - \mu c(x^k))$ while $\mathcal{L}$ is the Lagrangian function. A smaller and symmetric system can be obtained from Equation (2.6) by multiplying the last block row by $(X^k)^{-1}$ and adding it to the first block row

$$[\nabla^2\mathcal{L}_A + \Sigma^k]\Delta x^k = -\nabla\phi(x^k), \tag{2.8}$$

where Σ^k=${X^k}^{-1}V^k$. After solving Equation (2.8) for Δx^k, the step in the multipliers $\Delta\nu^k$ can be obtained using

$$\Delta\nu^k = \mu_{in}{X^k}^{-1}e - \nu^k - \Sigma^k\Delta x^k. \tag{2.9}$$

After the step directions are determined, the maximum step size $\alpha_x^{k,max}$ for primal variables and α_ν^k for dual variables can be calculated based on the fraction-to-the-boundary rule to ensure $x > 0$ and $\nu > 0$. With $\alpha_x^{k,max}$ as the initial guess, a line search is performed to get the step size α_x^k for primal variables. Finally, the values of primal and dual variable for the next interior-point iteration are calculated by

$$x_{k+1} = x^k + \alpha_x^k \Delta x^k \tag{2.10a}$$

$$\nu_{k+1} = \nu^k + \alpha_x^k \Delta \nu^k. \tag{2.10b}$$

2.2.3 Using PCG to Solve the Linear KKT System

The linear system (2.8) that needs to be solved at each interior-point iteration is

$$J^k \Delta x^k = -\nabla \phi(x^k), \tag{2.11}$$

where

$$J^k := \nabla^2 \mathcal{L}_A + \Sigma^k. \tag{2.12}$$

Solving this linear system is the dominant cost of the algorithm, and we want to use a PCG method since it can be parallelized on the GPU. Applying the PCG approach for solution of Equation (2.11) requires J^k to be P.D. If the original problem (1.1) is convex, J^k is guaranteed to be P.D. and PCG can be applied to the system.

For non-convex problems, if J^k is P.D., then PCG can be applied directly. If J^k is not P.D. (which can be detected in the PCG steps), then the PCG approach is aborted, a diagonal modifier $\delta_w I$ is added to the J^k and the linear system is solved again. If we continue to detect negative curvature the value of δ_w is increased according to the rule described in Wächter and Biegler (2006).

Forming J^k explicitly using sparse matrix-matrix multiplication can be expensive. At each iteration, PCG only requires a series of matrix-vector products with J^k.

Hence, in our implementation, J^k is not formed explicitly. Instead, the matrix vector products are performed across the right hand side expression in Equation (2.13).

$$J^k := \nabla^2 \mathcal{L}(x^k, \bar{\lambda} - \mu c(x^k)) + \mu A^k (A^k)^T + \Sigma^k + \delta_w I, \tag{2.13}$$

Therefore, each PCG step involves three sparse matrix-vector multiplications - $\nabla^2 \mathcal{L}$ with a vector, $[A^k]^T$ with a vector, and the multiplication of the resulting vector with A^k. This implicit implementation saves significant computational expense.

It is possible that the PCG method converges (based on its tolerance) even if the matrix J^k is not P.D. However, the starting point for Δx^k is set to zero, and the solution from the PCG approach is still guaranteed to be a descent direction for the barrier sub-problem (Dembo and Steihaug, 1983).

The PCG method can be accelerated with a suitable preconditioner. However it can be very challenging to efficiently implement many known preconditioners on a GPU since preconditioner factorization and backsolves are typically inefficient on the GPU (Li and Saad, 2013; Naumov, 2011). Furthermore, in our algorithm the matrix J^k is never explicitly formed, limiting the choice for preconditioner. Therefore, a simple diagonal preconditioner is used.

2.2.4 Algorithm Summary

This section summarizes the overall algorithm for solving problem (1.2). The algorithm is iterative at three levels. The first level replaces the original problem by a sequence of bound-constrained optimization problems using an augmented Lagrangian method. Each of these bound-constrained problems is solved using a nonlinear interior-point method. Inside the interior-point method, the barrier sub-problems are solved using a variation of Newton's method, and the linear system is solved iteratively with a preconditioned conjugate gradient method. In Algorithm 1 presented below, we provide the augmented Lagrangian method used in chapter. Following that, we present Algorithm 2 that describes the interior-point method used to solve the bound-constrained sub-problems in Algorithm 1.

```
Algorithm 1 : Augmented Lagrangian Method
1. Initialize
    Initialize the iteration index $j \leftarrow 0$.
    Set initial points $(x_0, \bar{\lambda_0})$ with $x_0 > 0$; overall convergence tolerance for the equality constraints and sub-problems $\eta_*$ and $\omega_*$; initial penalty parameter for $j$th iteration $\mu_j > 0$; initial tolerance for $j$th sub-problem $\omega_j$ and $\eta_j$.
2. Solve bound-constrained sub-problems
    Use Algorithm 2 to find a minimizer $(x_j^*, \nu_j^*)$ for (2.2) such that optimality error of the sub-problems $j$ satisfies $E_0(x_j^*, \nu_j^*) \leq \omega_j$ as computed in step 2 of Algorithm 2.
3. Update penalty parameter, Lagrangian multiplier and tolerance
    if $\| c(x_j) \| \leq \eta_j$ then
        if $\| c(x_j) \| \leq \eta_*$ and $E_0(x_j^*, \nu_j^*) \leq \omega_*$ then
            Stop with solution $(x^*, \nu^*, \lambda^*) \leftarrow (x_j^*, \nu_j^*, \bar{\lambda_j})$.
        end if
        Update multipliers $\bar{\lambda_j}$ and tighten tolerances $\eta_j$ and $\omega_j$.
    else
        Increase penalty parameter $\mu_j$ and update tolerance $\eta_j$ and $\omega_j$.
    end if
    Update $j \leftarrow j + 1$.
    Return to step 2.
```

The details about μ_j, ω_j, η_j initialization and how they are updated in step **3** are described in Conn et al. (1988). Now we provide the details of the interior-point algorithm to solve the jth sub-problem in step **2**.

Algorithm 2 : Interior-point Method

```
1. Initialize
   Initialize the iteration index $k \leftarrow 0$ and optimality tolerance $\omega_j$ from Algorithm 1.
   Set starting point $(x_0, \nu_0)$ with $\nu_0 > 0$; initial barrier parameter $\mu_{in} > 0$; tolerance constants $\kappa_\epsilon > 0$.
2. Check convergence for the bound-constrained sub-problem $j$
   if $E_0(x^k, \nu^k) \leq \omega_j$ then exit, solution found.
3. Check convergence for the barrier sub-problem
   if $E_{\mu_{in}}(x^k, \nu^k) \leq \kappa_\epsilon \mu_{in}$ then
      Update $\mu_{in}$.
      Repeat step 3 if $k$=0. Otherwise go to step 4.
   end if
4. Function Evaluations
   Evaluate $c(x^k)$, $\nabla_x f(x^k)$, $\nabla_x c(x^k)$, and $\nabla^2 \mathcal{L}(x^k, \bar{\lambda}' = \bar{\lambda} - \mu c(x^k))$.
5. Compute the search direction
   5.1 Solve (2.13) for $\Delta x^k$ using the PCG method on GPU. .
   5.2 Compute $\Delta \nu^k$ from (2.9). .
6. Backtracking line-search
   Calculate $\alpha^k_{x,max}$ and $\alpha^k_\nu$ based on the fraction-to-the-boundary rule.
   With $\alpha^k_{x,max}$ as the initial guess, perform a line search to obtain the step size $\alpha^k_x$.
7. Update iteration variables and continue to next iteration
   Compute $x^{k+1}, \nu^{k+1}$ with (2.10).
   Update $k \leftarrow k + 1$.
   Return to step 2.
```

The optimality error for the barrier problems used in steps **2** and steps **3** is calculated using

$$E_{\mu_{in}}(x^k, \nu^k) = \max\{ \parallel \nabla \mathcal{L}_A - \nu^k \parallel_\infty, \parallel X^k \nu^k - \mu_{in} e \parallel_\infty \}. \tag{2.14}$$

while for step **2**, μ_{in}=0.

2.2.5 Including General Variable Bounds

For the original problem formulation (1.1), we can generalize the above algorithm and transform the barrier sub-problems to

$$\min_{x\in\mathbb{R}^n} \quad \phi(x) = \mathcal{L}_A - \mu_{in}\sum_{i=1}^{n}\ln(x^{(i)} - x_L^{(i)}) - \mu_{in}\sum_{i=1}^{n}\ln(x_U^{(i)} - x^{(i)}). \tag{2.15}$$

Coupled with this, bound multipliers for both upper bounds and lower bounds ν_L and ν_U are introduced. Now, the barrier term of the Hessian is defined as $\Sigma^k = {S_L}^{-1}V_L + {S_U}^{-1}V_U$, with $S_L = \text{diag}(x^k - x_L)$ and $S_U = \text{diag}(x_U - x^k)$.

2.3 Parallel implementation

The authors implemented a serial version of Algorithm 1 and Algorithm 2 in C++ and compared the runtime of different components of the implementation. The performance of the serial implementation shows that in general about 80 percent of the runtime is spent on the linear solver, 18 percent of the runtime on function evaluations, and the rest on other calculations. Therefore, an effective parallel implementation must parallelize the PCG linear solver and the function evaluations. A discussion of the parallel implementation requires a brief introduction to the GPU architecture and CUDA programming. A typical NVIDIA GPU for scientific computing contains several Streaming Multiprocessors (SMs), each of which contains several CUDA cores. The memory architecture is complex. First, the GPU has global memory accessible to all SMs. Second, each SM has its own shared memory that is accessible to all CUDA cores on this specific SM. Finally, each CUDA core has its own register memory. Global memory is typically quite large (e.g. several GB), whereas shared memory and register memory are much smaller (e.g. 48 KB and 32 KB respectively on each SM). Furthermore, global memory has high latency while shared and register memory have significantly lower latency.

In November 2006, NVIDIA introduced CUDA, a parallel computing platform and programming framework for high performance computations. A CUDA program is composed of a host program to be executed on the CPU and one or more CUDA kernels for the GPU. A kernel is executed by a grid of thread blocks, each of which can contain hundreds of threads. This can result in thousands of concurrent threads. Threads within the same block are executed by the same SM in groups of 32 threads called warps. Each SM can execute several blocks concurrently.

Different levels of software optimization techniques can be applied including memory optimization, execution configuration optimization, and instruction optimization. Some of the most important software optimization techniques are coalesced and aligned global memory access. For example, for devices that have compute capability 2.0 and support double precision data, if all threads of the same half warp access adjacent blocks of global memory that are aligned at 128-byte boundary, only one memory transaction needs to be performed by the device for all threads. Other important software optimization techniques include minimizing data transfer between the host and the device, minimizing bank conflicts in shared memory, and grid size/block size optimization (occupancy optimization). Detailed descriptions about CUDA and performance optimization are presented in the Programming Guide (NVIDIA, 2011) and the Best Practices Guide (NVIDIA, 2012) provided by NVIDIA.

2.3.1 Parallel PCG on the GPU

In this section, we describe the parallel GPU implementation of the linear solver for sparse systems arising from the augmented Lagragian NLP algorithm. The main operations in the PCG method include: 1) solving the preconditioner equation, 2) vector-vector operations, and 3) matrix-vector multiplications. For a general preconditioner, one would typically need a backsolve which is, in general, inefficient when applied on a GPU. However, since we are using a diagonal preconditioner, the backsolve becomes a vector-vector operation. Vector-vector operations are straightforward to implement on the GPU. For example, to add two vectors, each element addition is performed on a separate thread. A slightly more challenging vector-vector operation worth highlighting is the dot product because it involves a reduction operation from all CUDA cores. The details about an efficient implementation of this operation can be found in (Harris, 2007). Matrix-vector operations are more complex, but also have a larger scope for optimization, and we discuss these in the following sub-sections.

2.3.1.1 Sparse Matrix-Vector Multiplication

The most expensive step in the PCG method is the sparse matrix-vector multiplication (SpMV). The challenge when implementing a general parallel SpMV operation is that the data access is irregular for an unstructured matrix. As mentioned before, coalesced and aligned global memory access is fast while irregular global memory access is slow for a GPU. Consequently, optimizing memory access by choosing the best sparse matrix storage format for a particular matrix structure is critical for improving performance. For this purpose, we evaluate four different sparse matrix storage formats.

The coordinate (COO) format employs three arrays to store the row coordinates, column coordinates, and values of every nonzeros. Typically, nonzeros are stored row-wise, and to parallelize SpMV using the COO format, one thread is assigned for each of the nonzeros to perform the multiplication. Then the sum of threads of the same row is calculated by performing segmented reduction. The performance of SpMV with COO format is generally poor but consistent across different sparse matrix structures because of its fine granularity.

The compressed Sparse Row (CSR) format stores the values and column indices of the nonzero values in two arrays ordered row-wise. It uses a third vector to store the starting nonzero position of each row. An intuitive way of implementing SpMV for the CSR format on a GPU is to assign one thread for each of the nonzeros. However, this implementation, called a *scalar* kernel, makes the data access of contiguous threads far from each other when the number of nonzeros per row is larger than one. A more sophisticated approach used by CUSPARSE (Bell and Garland, 2009), called the *vector* kernel, assigns one warp for each row, while IBM's SpMV library (Baskaran and Bordawekar, 2008) shows it is more efficient to assign a half warp for each row. The results of the half warps are then saved in shared memory and summed using parallel reduction. The advantage of this kernel is that the threads of the same half warp access data contiguously. Although CSR works well when the number of nonzeros in each row is larger than 16, its performance is poor when number of nonzeros per row is small, causing some threads of the half warp to remain idle.

The Ellpack (ELL) format assumes that the number of nonzeros in each row is constant and rows with fewer nonzeros are zero-padded. Since the number of nonzeros per row is fixed, it uses only two arrays to store the column indices and values of each of the nonzeros. To parallelize SpMV for the ELL format, one thread is assigned for each row. The i^{th} nonzeros in each row are stored contiguous, so the data access across contiguous threads is coalesced. However, when the number of nonzeros vary dramatically over the rows, the ELL format suffers from requiring too many zeros to pad the rows, hence it will not only cause extra computational work but also extra memory.

The Hybrid (HYB) format combines the efficiency of the ELL format for structured problems and the stability of COO format over different sparse structures. The first K entries of each row are stored in the ELL format while the remaining entries are stored in the COO format. The value of K is selected to ensure that the majority of the data is stored in the ELL parts.

Taking advantage of the discussion above about these sparse matrix formats and their implementation on the GPU, we now discuss the implementation details of PCG on GPU.

For the vector-vector operations performed in PCG, we make use of the CUBLAS library (Toolkit, 2011), which is an implementation of dense BLAS (Basic Linear Algebra Subprograms) on a GPU. However, within the PCG implementation, two results of the dot product need be transferred back from the GPU to the CPU at each iteration in order to determine whether negative curvature has been encountered and whether the PCG iterations have converged. While the data transfer for these quantities can be time consuming, the GPU used in this chapter can perform data transfer between pinned host memory and global memory concurrently with device computations. We can take advantage of this by overlapping the data transfer with the SpMV kernel execution for the next step. Since the CUBLAS 4.1 library currently does not implement this overlapping feature, we have written a custom kernel for the dot product.

For SpMV, there are already several libraries to choose from. The CUSPARSE library (Bell and Garland, 2009) has implemented sparse matrix-vector multiplication with different sparse matrix formats. Recall that we do not form J^k explicitly, but rather implement the required matrix-vector products by multiplying across the expression in Equation (2.13). For $[A^k]^T$ and $\nabla^2\mathcal{L}$, we have selected the HYB format implemented in the CUSPARSE library since numerical results in (Bell and Garland, 2009) show that HYB is the fastest format for a majority of unstructured matrices. We selected the CSR format for A^k because A^k has few rows but a large number of nonzeros per row. Since the existing library does not support the overlap between data transfer and kernel execution, we have written our own CSR kernel for the SpMV.

2.3.2 Parallelize Function Evaluations

As we will show later in Section 4, for the serial version of the algorithm the time spent on PCG iterations is around 65-85 percent of the runtime. Therefore the maximum speedup achievable throughout parallelization of the PCG steps is only about 3 to 7 times. The majority of the remaining runtime is spent on evaluating the Hessian of Lagrangian function, gradient of the objective function, gradient of the equality constraints, residual of the equality constraints, and the objective function. If these function evaluations are also parallelized, a several fold improvement can be further expected. However, so far, no library exists for automatic parallel function evaluations of general NLP problems on the GPU, and the development of a general library for this task is challenging even on the CPU. Therefore, we developed problem specific code for parallel function evaluations on the GPU for each test problem presented in our results. The purpose of further parallelizing function evaluations is to highlight the potential of this algorithm on the GPU instead of providing a general solution.

2.4 Numerical Results

In this section, we present the numerical performance results of the proposed algorithm on selected problems from the COPS test set. All the test problems are written in the AMPL modeling language(Gay and Kernighan, 2002). Since the augmented Lagrangian method is more efficient for problems with few constraints, the following six problems are selected - torsion, bearing, minsurf, lane-emden, dirichlet, and henon. The first three problems have no equality constraints, while the last three problems have few equality constraints relative to the number of variables. All test problems have bound constraints on all variables. The number of variables for the selected problems ranges from 10,000 to 120,000. For comparison, the state-of-the-art nonlinear solver Ipopt with MA27 from the Harwell Subroutine Library is also used to solve these problems. For both our algorithm and Ipopt, we set the convergence tolerances for both equality constraints and optimality of the sub-problems to 10^{-6}. Both solvers are evaluated using a 2.50GHz Intel Xeon E5420 quad-core CPU and a Tesla C2050 GPU with CUDA driver version 4.2. The Tesla C2050 contains 14 SMs (each contains 32 cores) and a total of 3GB memory.

Before we parallelize the proposed algorithm, we need to make sure the serial augmented Lagrangian method is competitive when compared with existing solvers. The proposed algorithm was implemented in C++, and the timing results are compared with those from Ipopt. Figure 2.1 shows that although the serial algorithm is in general slower than Ipopt, the runtime ratio is below 2.5 for these test problems. Table 2.1 shows the wall-clock time for each of these problems.

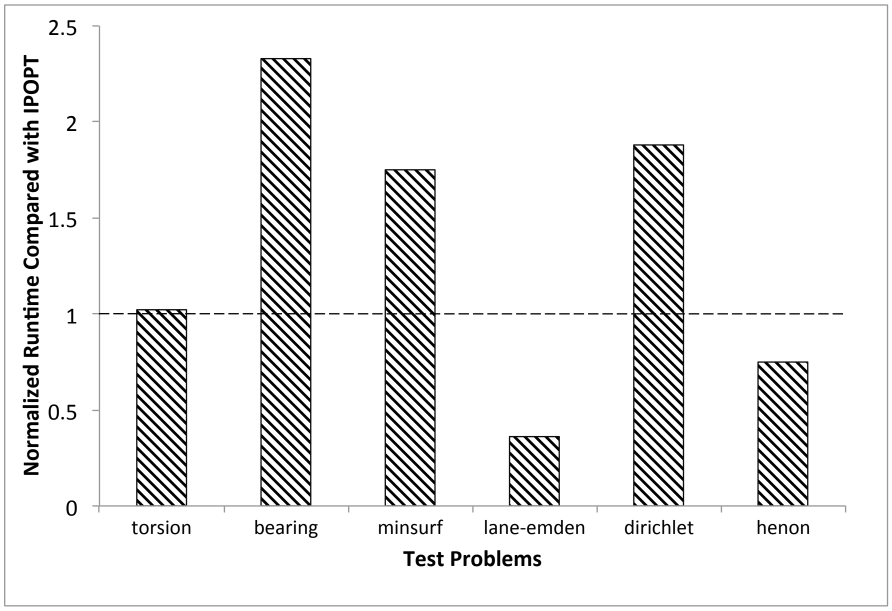


Figure 2.1.: Runtime of the serial augmented Lagrangian interior-point method normalized with respect to Ipopt (– –).

2.4.1 Performance of the Parallel Code

Figure 2.2 shows the average performance of the parallel PCG implementation inside the augmented Lagrangian algorithm. As shown in this figure, each parallel PCG iteration performs on average 10-21 times faster than the serial implementation. Because of different architectures and round-off errors, the number of iterations taken by the parallel implementation can be different from the serial implementation. Between the two implementations, the number of barrier iterations never differ more than 10 percent for all test problems, and the number of PCG iterations never differ more than 60 percent. Therefore, the time per PCG iteration is used to compare the serial code and the parallel code.

Once the PCG method is parallelized, the time for function evaluations becomes the performance bottleneck. For example, Figure 2.3 depicts the time composition

Table 2.1: Wall-clock time of the serial algorithm for selected test problems.

Problem	n	m	IPOPT(s)	Serial AL-IP(s)
torsion	120000	0	21.27	21.69
bearing	120000	0	20.46	47.7
minsurf	50000	0	115.4	202.28
lane-emden	19322	81	397.67	144.54
dirichlet	18042	81	279.03	523.87
henon	10882	81	2628.87	1970.7

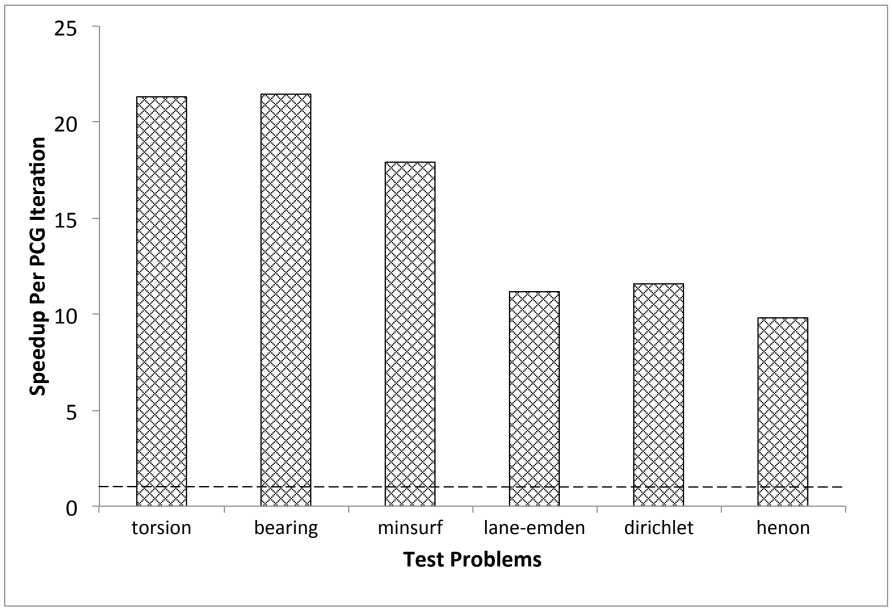


Figure 2.2.: Average speedup of each PCG iteration comparing parallel PCG implementation with serial PCG implementation (– –).

of the serial code and the parallel PCG code for test problem dirichlet. The PCG linear solver takes more than 80 percent of the total runtime in the serial code but only takes less than 30 percent in the parallel code. More than 90 percent of the remaining runtime is spent on the function evaluations, which motivates performing the function evaluations in parallel as well.

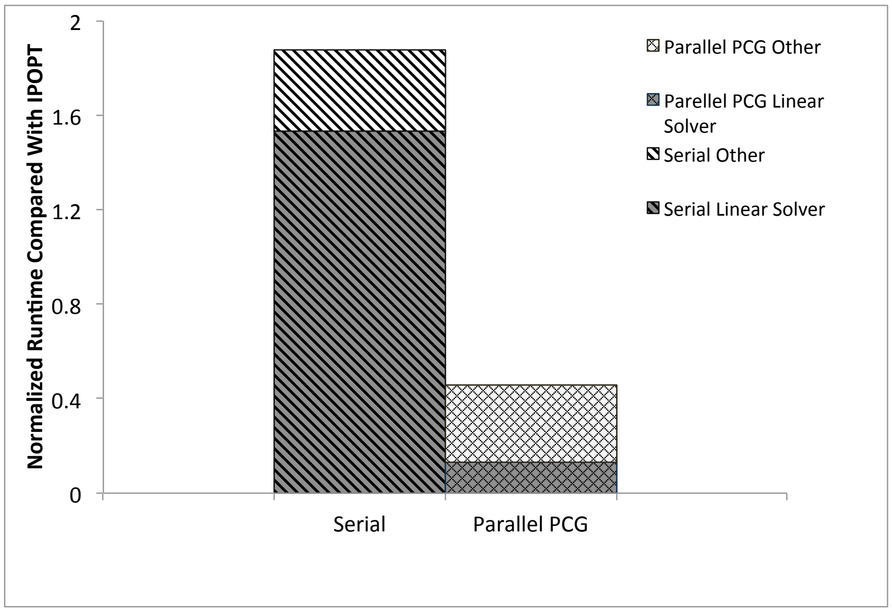


Figure 2.3.: Runtime composition of serial algorithm and parallel PCG algorithm for problem Dirichlet.

Figure 2.4 gives the overall speedup on all test problems with only the PCG step parallelized, and with both PCG and function evaluations parallelized. If only the PCG steps are parallelized, the algorithm obtains a speedup of 3-6, whereas, parallelizing the function evaluations pushes the overall speedup to 13-18.

2.5 Concluding Remarks

In summary, we have developed a parallel augmented Lagrangian interior-point algorithm for solving large-scale NLP problems using graphics processing units. The augmented Lagrangian approach ensures that the KKT system is positive definite for convex problems. This enables us to solve the KKT system using a parallel PCG

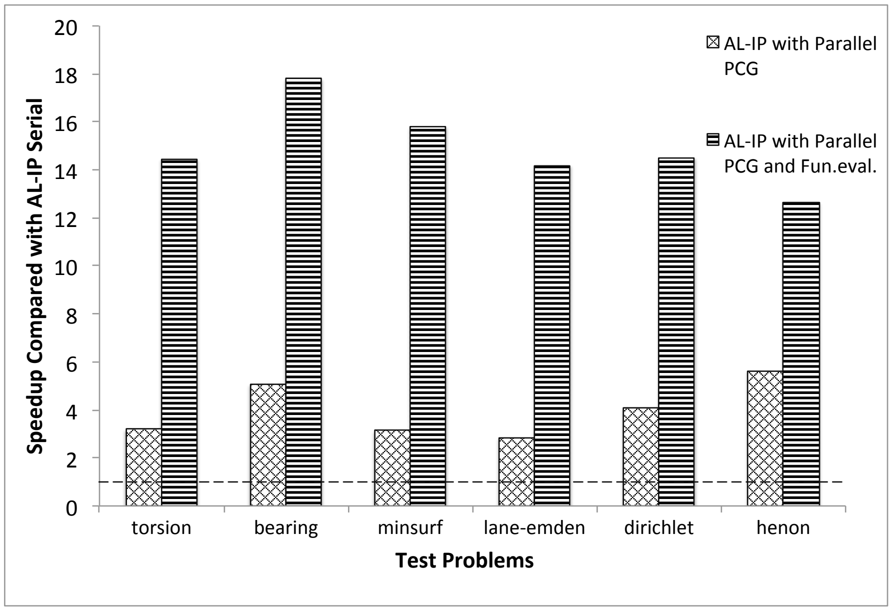


Figure 2.4.: Speedup with only PCG step parallelized and with both PCG and function evaluation parallelized with respect to serial implementation (– –).

method on the GPU. An overall speedup of 13-18 was obtained on six test problems from COPS test set.

The GPU we use (Tesla C2050) has 448 cores, 515 Gflops double precision floating point performance, 144 GB/sec memory bandwith and around 3 GB global memory. However, with the rapid development of parallel platforms, the latest GPUs like the Tesla K20X already have 2688 cores,1.31 teraflops double precision floating point performance, 250 GB/sec memory bandwith and around 6 GB global memory, which can push the performance of our GPU implementation further.

3. EXPLICIT SCHUR COMPLEMENT METHOD FOR STOCHASTIC PROGRAMS

This chapter describes the parallel Schur complement method used in solving the stochastic programs of the form (1.6) with distributed computing clusters or multi-core machines. We will start with the general theories regarding interior-point methods for solving general NLP problems of the form (1.2) in Section 3.1. The details of interior-point algorithms can be found in Wächter (2002) and Wächter and Biegler (2006). For continuous nonlinear optimization problems, interior-point methods, sequential quadratic programming (SQP) methods, and augmented Lagrangian methods are the most successful general purpose algorithms (Nocedal and Wright, 1999). Several interior-point implementations exist, including Ipopt(Wächter and Biegler, 2006), LOQO (Vanderbei and Shanno, 1999), and KNITRO/DIRECT (Waltz et al., 2006); SQP implementations include SNOPT (Gill et al., 2002), FILTER-SQP (Fletcher and Leyffer, 2002), and KNITRO/ACTIVE (Byrd et al., 2003); and augmented Lagrangian methods have been implemented in MINOS (Murtagh and Saunders, 1982) and Lancelot (Conn et al., 1988).

For structured NLP problems, particularly stochastic programs, interior-point methods are preferable because the structure of the linear system used to compute the step remains the same at each iteration, making the development of tailored linear solvers appropriate. The linear systems derived using interior point methods for stochastic programming problems have the block-bordered-diagonal form. Currently, all the well-known parallel interior-point solvers for these NLP problems (e.g. OOPS (Gondzio and Grothey, 2009), Schur-IPOPT (Kang et al., 2014), and PIPS-NLP (Chiang et al., 2014)) are based on the parallel implementation of Schur complement method of the KKT system. This approach has almost perfect strong scaling efficiency for solving the KKT system when the number of first stage variables is small. One disadvantage of this approach is that forming and solving the dense Schur system become the bottleneck when the number of first stage variables is large. This disadvantage can be overcome by several methods discussed in Section 1.4 and the algorithm proposed in Chapter 4.

3.1 Interior-Point Method for General NLP Problems

In this section, we present an interior-point algorithm to deal with nonlinear programming problems of the form (1.2). It is quite similar to the algorithm introduced in Section 2.2.2. The only difference is that the nonlinear programming problems discussed in this section also includes equality constraints. Necessary modifications to handle the general form (1.1) have already been discussed In Section 2.2.5.

An interior-point method solves the problem (1.2) by solving a sequence of barrier subproblems of the form:

$$\min_{x \in \mathbb{R}^n} \ \varphi(x) = f(x) - \mu_{in} \sum_{i=1}^{n} \ln(x^{(i)}) \tag{3.1a}$$

$$\text{s.t. } \ c(x) = 0, \tag{3.1b}$$

where $x^{(i)}$ denotes the ith component of the vector x, and $\mu_{in} > 0$ is the barrier parameter.

The first-order optimality conditions of the barrier sub-problem are

$$\nabla f(x) - \mu_{in} X^{-1} e + \nabla_x c(x) \lambda = 0 \tag{3.2a}$$

$$c(x) = 0 \tag{3.2b}$$

where X=diag(x), and e is a vector with all elements equal to 1. The optimality conditions also have an implicit constraint of $x \geq 0$. The primal-dual reformulation can be written by introducing $\nu = \mu_{in} X^{-1} e$,

$$\nabla f(x) - \nu + \nabla c(x)\lambda = 0 \tag{3.3a}$$

$$c(x) = 0 \tag{3.3b}$$

$$X\nu - \mu_{in} e = 0. \tag{3.3c}$$

When μ_{in} is 0, the above equations together with $x \geq 0, \nu \geq 0$ are the KKT conditions for the problem (1.2). We can use a variation of Newton's method to solve the primal-dual system of Equations (3.3). At each iteration k of the Newton's method, the step direction is calculated by solving the linear system

$$\begin{bmatrix} H^k & A^k & -I \\ (A^k)^T & 0 & 0 \\ V^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta \lambda^k \\ \Delta \nu^k \end{bmatrix} = - \begin{bmatrix} \nabla f(x^k) + A^k \lambda^k - \nu^k \\ c(x^k) \\ X^k \nu^k - \mu_{in} e \end{bmatrix}. \tag{3.4}$$

Here H^k denotes the Hessian of the Lagrangian function $\nabla^2 \mathcal{L}$, $A^k := \nabla c(x^k)$, while Δx^k and $\Delta \lambda^k$, $\Delta \nu^k$ are the search directions in x, λ and ν respectively. The Lagrangian function is of the form:

$$\mathcal{L}(x, \lambda, \nu) = f(x) + \lambda^T c(x) - \nu^T x. \tag{3.5}$$

A smaller and symmetric system can be obtained from Equation (3.4) by multiplying the last block row by $(X^k)^{-1}$ and adding it to the first block row

$$\begin{bmatrix} W_k & A^k \\ (A^k)^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta \lambda^k \end{bmatrix} = - \begin{bmatrix} \nabla \varphi(x^k) + A^k \lambda^k \\ c(x^k) \end{bmatrix}, \tag{3.6}$$

where W_k=$H^k + \Sigma^k$ and Σ^k=$(X^k)^{-1}V^k$. After solving the Equation (3.6) for Δx^k, $\Delta\nu^k$ can be obtained by using

$$\Delta\nu^k = \mu_{in}(X^k)^{-1}e - \nu^k - \Sigma^k\Delta x^k. \tag{3.7}$$

After the step directions are determined, the maximum step size $\alpha_x^{k,max}$ for primal variables and α_ν^k for dual variables can be calculated based on the fraction-to-the-boundary rule to ensure $x > 0$ and $\nu > 0$. The step size α_x^k is computed using a line search filter method with $\alpha_x^{k,max}$ as the initial guess. The basic idea of this method is to find a step size making sufficient progress in either decreasing $\varphi(x)$ or decreasing the violation of $c(x)$. Finally, the values of primal and dual variables for the next interior-point iteration are calculated using the Equation (2.10).

One requirement for the line search filter method to guarantee a certain descent property is that a projection of W into the null space N of $(A^k)^T$ must be positive definite (Wächter and Biegler, 2005). Given a matrix M, its inertia denoted by In(M), is the integer triple indicating the number of positive, negative and zero eigenvalues (Forsgren et al., 2002). Therefore the line search filter method requires In$(N^TWN) = (n-m, 0, 0)$.

For the KKT matrix

$$K = \begin{bmatrix} W_k & A^k \\ (A^k)^T & 0 \end{bmatrix} \tag{3.8}$$

Forsgren et al. (2002) shows that In(K) = In$(N^TWN) + (m, m, 0)$ when A has full rank. As a consequence, under the assumption of linear independence constraint qualification (LICQ), In$(K) = (n, m, 0)$ if and only if N^TWN is positive definite. Therefore, the requirement of line search filter method will be satisfied if the following condition holds

$$\text{In}(K) = (n, m, 0). \tag{3.9}$$

The information of inertia can be obtained as a byproduct of LDL factorization. In the case when the conditions (3.9) does not hold, inertia correction can be performed by a diagonal modification of the KKT matrix (Wächter and Biegler, 2006). The modified KKT matrix is of the form:

$$K = \begin{bmatrix} W_k + \delta_w I & A^k \\ (A^k)^T & -\delta_c I \end{bmatrix}, \tag{3.10}$$

where δ_w and δ_c are two positive values.

3.2 Schur Complement Method for Stochastic Programs

The dominant computational cost of the interior point method is the solution of Equation (3.6). For stochastic programs, the problem structure can be exploited to develop a tailored parallel linear solver. In this section, instead of solving the original stochastic programs of the form (1.6), we solve the equivalent problem (3.11) by duplicating the first stage variables x_0 as $x_{0,s}$, $s \in \mathcal{S}$

$$\min\ f_0(x_{0,1}) + \sum_{s\in\mathcal{S}} f_s(x_s, x_{0,s}) \tag{3.11a}$$

$$\text{s.t.}\ \ c_0(x_{0,1}) = 0 \qquad (\lambda_0) \tag{3.11b}$$

$$c_s(x_s, x_{0,s}) = 0 \qquad (\lambda_s),\ \ s \in \mathcal{S} \tag{3.11c}$$

$$x_{0,1} \geq 0 \qquad (\nu_0) \tag{3.11d}$$

$$x_s \geq 0 \qquad (\nu_s)\ \ s \in \mathcal{S}. \tag{3.11e}$$

$$x_{0,s} = x_0 \qquad (\sigma_s)\ \ s \in \mathcal{S}. \tag{3.11f}$$

Here, the equality and bound contraints previously applied on x_0 only transfer to that of $x_{0,1}$ to prevent redundant constraints.

Without the Equation (3.11f), the above formulation can be decomposed into S sub-problems. The subproblem 1 has the form

$$\min_{x_1,x_{0,1}} f_0(x_{0,1}) + f_s(x_1, x_{0,1}) \tag{3.12a}$$

$$\text{s.t. } c_0(x_{0,1}) = 0 \tag{3.12b}$$

$$c_1(x_1, x_{0,1}) = 0 \tag{3.12c}$$

$$x_{0,1} \geq 0 \tag{3.12d}$$

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

with the Lagrangian function of subproblem 1 defined as

$$\begin{aligned}\mathcal{L}_1(x_{0,1}, x_1, \lambda_1, \lambda_0, \nu_1, \nu_0) =& f_0(x_{0,1}) + f_1(x_1, x_{0,1}) + {\lambda_1}^T c_1(x_{0,1}, x_1) \\ &+ {\lambda_0}^T c_0(x_{0,1}) - \nu_1^T x_1 - \nu_0^T x_{0,1}\end{aligned} \tag{3.13}$$

Each subproblem s, $s \in \{2..S\}$ is of the form

$$\min_{x_s,x_{0,s}} \sum_{s\in\mathcal{S}} f_s(x_s, x_{0,s}) \tag{3.14a}$$

$$\text{s.t. } c_s(x_s, x_{0,s}) = 0 \tag{3.14b}$$

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

with the the Lagrangian function defined as:

$$\mathcal{L}_s(x_{0,s}, x_s, \lambda_s, \nu_s) = f_s(x_s, x_{0,s}) + {\lambda_s}^T c_s(x_{0,s}, x_s) - \nu_s^T x_s \tag{3.15}$$

The Lagrangian of the whole problem (3.11) can be formulated as:

$$\mathcal{L}(x, \lambda, \nu, \sigma) = \sum_{s\in\mathcal{S}} \mathcal{L}_s + \sigma_s^T (x_{0,s} - x_0) \tag{3.16}$$

For the problem (3.11), System (3.6) has the following arrowhead form after reformulation

$$\begin{bmatrix} K_1 & & & & B_1 \\ & K_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & K_S & B_S \\ B_1^T & B_2^T & \dots & B_S^T & K_0 \end{bmatrix} \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_S \\ \Delta w_0 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_S \\ r_0 \end{bmatrix}, \tag{3.17}$$

where,

$$
\begin{aligned}
\Delta w_0^T &:= [\Delta x_0^T] \\
\Delta w_1^T &:= [\Delta x_1^T, \Delta x_{0,1}{}^T, \Delta \lambda_1^T, \Delta \lambda_0^T, \sigma_1^T] \\
\Delta w_s^T &:= [\Delta x_s^T, \Delta x_{0,s}{}^T, \Delta \lambda_s^T, \sigma_s^T] && \forall s \in \{2..S\} \\
r_0^T &:= \sum_{s \in \mathcal{S}} \sigma_s \\
r_1^T &= -\left[\left(\nabla_{x_1} \mathcal{L}_1 + \nu_1 - \mu_{in} X_1{}^{-1} e \right)^T, c_1^T, c_0^T, (x_{0,1} - x_0)^T \right] \\
r_s^T &= -\left[\left(\nabla_{x_s} \mathcal{L}_s + \nu_s - \mu_{in} X_s{}^{-1} e \right)^T, c_s^T, (x_{0,s} - x_0)^T \right] && \forall s \in \{2..S\} \\
K_0 &:= \begin{bmatrix} 0_{n_0} \end{bmatrix} \\
K_1 &:= \begin{bmatrix} W_1 & H_{0,1}^T & A_1 & A_0 & 0 \\ H_{0,1} & W_{0,1} & T_1 & 0 & I \\ A_1^T & T_1^T & 0 & 0 & 0 \\ A_0^T & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 \end{bmatrix} \\
K_s &:= \begin{bmatrix} W_s & H_{0,s,s}^T & A_s & 0 \\ H_{0,s,s} & W_{0,s} & T_s & I \\ A_s^T & T_s^T & 0 & 0 \\ 0 & I & 0 & 0 \end{bmatrix} && \forall s \in \{2..S\} \\
B_1 &:= \begin{bmatrix} 0 & 0 & 0 & 0 & -I \end{bmatrix} \\
B_s &:= \begin{bmatrix} 0 & 0 & 0 & -I \end{bmatrix} && \forall s \in \{2..S\} \\
W_s &:= H_s + X_s^{-1} V_s && \forall s \in \{1..S\} \\
W_{0,1} &:= H_{0,1} + X_{0,1}^{-1} V_{0,1} \\
W_{0,s} &:= H_{0,s} && \forall s \in \{2..S\}
\end{aligned}
\tag{3.18}
$$

Here $c_s{=}c_s(x_s, x_{0,s})$, $A_s{=}\nabla_{x_s} c_s(x_s, x_{0,s})$, $T_s{=}\nabla_{x_{0,s}} c_s(x_s, x_{0,s})$, $H_s{=}\nabla^2_{x_s x_s} \mathcal{L}_s$, $H_{0,s}{=}\nabla^2_{x_{0,s} x_{0,s}} \mathcal{L}_s$, $H_{0,s,s}{=}\nabla^2_{x_{0,s} x_s} \mathcal{L}_s$.

Assuming that all K_s are of full rank, we can show with the Schur complement method that the solution of the Equation (3.17) is equivalent to that of the following system

$$\underbrace{(K_0 - \sum_{s\in\mathcal{S}} B_s^T K_s^{-1} B_s)}_{:=Z}\Delta w_0 = \underbrace{r_0 - \sum_{s\in\mathcal{S}} B_s^T K_s^{-1} r_s}_{:=r_Z} \tag{3.19a}$$

$$K_s \Delta w_s = r_s - B_s \Delta w_0, \quad \forall s \in \mathcal{S}. \tag{3.19b}$$

It can also been shown that the inertia information of the whole KKT matrix K can be derived from the inertia of Z and K_s (Kang et al., 2014)

$$\mathrm{In}(K) = \sum_{s\in\mathcal{S}} \mathrm{In}(K_s) + \mathrm{In}(Z). \tag{3.20}$$

Therefore, the inertia correction can still be performed using the Schur complement method to satisfy the requirements of line-search filter method.

The system (3.19) can be solved with 3 steps. The first step is to form Z and r_Z by adding the contribution from each block. This step requires the factorizations of one sparse matrix K_1 of size $n_1 + 2n_0 + m_1 + m_0$ and $S - 1$ sparse matrix K_s of size $n_s+2n_0+m_s$. Besides a total of S factorizations of block matrix, this step also requires a total of $(S + 1)n_0$ backsolves. The second step is to solve the Equation (3.19a) to get direction of the first stage variables Δw_0. This step requires one factorization and one backsolve of the dense matrix Z. With Δw_0, the third step is to compute Δw_s from Equation (3.19b). This step requires a total of S backsolves of the block spase matrix.

The Schur complement decomposition in (3.19) restricts the pivot sequences that are possible and is rarely beneficial over (3.17) in serial if optimal ordering is used. However, finding the optimal ordering itself is an NP hard problem. In many linear solver, some heuristics are used in the ordering algorithms. Therefore, in serial, it is hard to predict which system is faster to solve. However, one significant advantage of solving the system (3.19) is that both step 1 and step 3 can be easily parallelized. When n_0 is relatively small, and thus the cost of factorizing matrix Z in step 2 is negligible, the efficiency of the parallel implementation can be very close to 1. Another advantage of using the parallel Schur complement method on distributed architectures is that the memory requirement is much smaller for each node than solving the system (3.6).

The disadvantage of this approach is that as the number of first stage variables increases, the cost of forming the Schur complement in step 1 increases linearly, and the cost of factorization of the (possibly) dense matrix Z increases cubically. Therefore, this method is not appropriate to be directly applied to problems with a large number of first stage variables.

3.3 Remarks

We close this chapter by discussing the advantages and disadvantages of the formulation (3.11) over the formulation (1.6). Formulation (3.11) duplicates the first stage variables for each scenario. The interior point method for the formulation (1.6) is not derived in this dissertation, but the QP version of the formulation (1.6) is considered in Chapter 4.

One advantage of using the formulation (3.11) is that the Schur complement matrix is P.D. if the original KKT system and each K_s block has the correct Inertia. This property enables use of a PCG procedure to solve the Schur system (Kang et al., 2014). This approach avoids both the explicit formation and factorization of the dense Schur complement matrix.

Another advantage of using formulation (3.11) is that it facilitates the software development process. The Equation (3.18) and (4.9) indicate that the KKT system of the whole problem can be constructed by the Jacobian, Hessian, and function evaluations of subblocks for both formulations. In other words, The whole model can be constructed by generating one model file (e.g. AMPL file) for each subblock and setting appropriate suffixes in each model file to identify first stage variables. Therefore, the model evaluation can be performed in parallel. The specialty of formulation (3.11) is that the Hessian and Jacobian for the subblocks can be directly used. For example, the Jacobian evaluated for subproblem s, $s \in \{2..S\}$, is $\nabla_{x_s,x_{0,s}} c_s(x_s, x_{0,s})^T = [A_s^T, T_s^T]$. For the formulation (3.11), $\nabla_{x_s,x_{0,s}} c_s(x_s, x_{0,s})^T$ can be used directly in Equation (3.18). However, For the formulation (1.6), it must be split into A_s^T and T_s^T in Equation (4.9).

The final advantage of using the formulation (3.11) is that it has a smaller Schur complement matrix at the cost of larger sparse K_s matrices. Using the formulation (3.11), the size of Z is n_0, the dimension of K_1 is $n_1 + 2n_0 + m_1 + m_0$, and that of K_s, $s \in \{2..S\}$, is $n_s + 2n_0 + m_s$. For the formulation (4.9), the size of Z is $n_0 + m_0$ and the dimension of K_s, $s \in \mathcal{S}$, is $n_s + m_s$. Thus, formulation (3.11) has a lower computational cost of factorizing the Schur complement but a higher computational cost of forming the Schur complement. The cost of factorizing the Schur complement increases much faster than that of forming the Schur complement as the dimension of first stage increases. Therefore, when the dimension of first stage is large, the formulation (3.11) performs better, although explicit Schur complement method is no longer a good choice in this circumstance.

4. CLUSTERING-BASED PRECONDITIONING FOR STOCHASTIC PROGRAMS[1]

Chapter 3 describes an explicit Schur complement method to solve stochastic programs in parallel. One drawback of an straightforward implementation of this method is that when the dimension of first stage variables is large, formation and factorization of Schur complement becomes the bottleneck. In this chapter, we discuss one method that can solve stochastic programs with a large number of first stage variables in parallel efficiently. The distinction of this method with all other parallel interior-point solvers for stochastic programs is that this method is not based on Schur complement decomposition, although Schur complement is used in deriving mathematical properties of this approach.

This method uses a clustering-based preconditioning strategy for KKT systems. The key idea is to perform adaptive clustering of scenarios (inside-the-solver) based on their influence on the problem as opposed to cluster scenarios based on problem data alone, as is done in existing (outside-the-solver) approaches. We derive spectral and error properties for the preconditioner and demonstrate that scenario compression rates of up to 94% can be obtained, leading to dramatic computational savings. In addition, we demonstrate that the proposed preconditioner can avoid scalability issues of Schur complement decomposition in problems with large first-stage dimensionality.

[1]Part of this section is reprinted with permission from "Clustering-Based Preconditioning for Stochastic Programs" by Cao, Y., Laird, C.D., and Zavala, V. M., 2015. Submitted to Computational Optimization and Applications.

4.1 Preliminaries

We consider two-stage stochastic programs of the form

$$\min \left(\frac{1}{2}x_0^T H_0 x_0 + d_0^T x_0\right) + \sum_{s\in\mathcal{S}} \xi_s \left(\frac{1}{2}x_s^T H_s x_s + d_s^T x_s\right) \tag{4.1a}$$

$$\text{s.t.} \qquad A_0^T x_0 = b_0, \quad (\lambda_0) \tag{4.1b}$$

$$T_s^T x_0 + A_s^T x_s = b_s, \quad (\lambda_s),\ s \in \mathcal{S} \tag{4.1c}$$

$$x_0 \geq 0, \quad (\nu_0) \tag{4.1d}$$

$$x_s \geq 0, \quad (\nu_s),\ s \in \mathcal{S}. \tag{4.1e}$$

The problem variables are $x_0, \nu_0 \in \Re^{n_0}$, $x_s, \nu_s \in \Re^{n_s}$, $\lambda_0 \in \Re^{m_0}$, and $\lambda_s \in \Re^{m_s}$. The total number of variables is $n := n_0 + \sum_{s\in\mathcal{S}} n_s$, of equality constraints is $m := m_0 + \sum_{s\in\mathcal{S}} m_s$, and of inequalities is n. We refer to (x_0, λ_0, ν_0) as the first-stage variables and to (x_s, λ_s, ν_s), $s \in \mathcal{S}$, as the second-stage variables. We refer to Equation (4.1a) as the cost function. The *data* defining problem (4.1) is given by the cost coefficients d_0, H_0, H_s, d_s, the right-hand side coefficients b_0, b_s, and the matrix coefficients T_s, A_s. We refer to H_s, d_s, b_s, T_s, A_s as the *scenario data*. We define scenario probabilities as $\xi_s \in \Re_+$ but we drop them from the notation by redefining $H_s \leftarrow \xi_s H_s$ and $d_s \leftarrow \xi_s d_s$.

As is typical in stochastic programming, the number of scenarios can be large and limits the scope of existing off-the-shelf solvers. In this chapter, we present strategies that cluster scenarios at the linear algebra level to mitigate complexity.

We start the discussion by presenting some basic notation. The Lagrange function of (4.1) is given by

$$\begin{aligned}\mathcal{L}(x, \lambda, \nu) = \frac{1}{2}x_0^T H_0 x_0 + d_0^T x_0 + \lambda_0^T (A_0^T x_0 - b_0) - \nu_0^T x_0 \\ + \sum_{s\in\mathcal{S}} \left(\frac{1}{2}x_s^T H_s x_s + d_s^T x_s + \lambda_s^T (T_s^T x_0 + A_s^T x_s - b_s) - \nu_s^T x_s\right).\end{aligned} \tag{4.2}$$

Here, $x^T := [x_0^T, x_1^T, ..., x_S^T]$, $\lambda^T := [\lambda_0^T, \lambda_1^T, ..., \lambda_s^T]$, and $\nu^T := [\nu_0^T, \nu_1^T, ..., \nu_S^T]$. In a primal-dual interior-point (IP) setting we seek to solve nonlinear systems of the form

$$\nabla_{x_0}\mathcal{L} = 0 = H_0 x_0 + d_0 + A_0\lambda_0 - \nu_0 + \sum_{s\in\mathcal{S}} T_s\lambda_s \tag{4.3a}$$

$$\nabla_{x_s}\mathcal{L} = 0 = H_s x_s + d_s + A_s\lambda_s - \nu_s,\ s\in\mathcal{S} \tag{4.3b}$$

$$\nabla_{\lambda_0}\mathcal{L} = 0 = A_0^T x_0 - b_0 \tag{4.3c}$$

$$\nabla_{\lambda_s}\mathcal{L} = 0 = T_s^T x_0 + A_s^T x_s - b_s,\ s\in\mathcal{S} \tag{4.3d}$$

$$0 = X_0 V_0 e_{n_0} - \mu_i n e_{n_0} \tag{4.3e}$$

$$0 = X_s V_s e_S - \mu_i n e_{n_s},\ s\in\mathcal{S}, \tag{4.3f}$$

with the implicit condition $x_0, \nu_0, x_s, \nu_s \geq 0$. Here, $\mu_i n \geq 0$ is the barrier parameter and $e_{n_0} \in \Re^{n_0}, e_{n_s} \in \Re^{n_s}$ are vectors of ones. We define the diagonal matrices $X_0 := \text{diag}(x_0), X_s := \text{diag}(x_s)$, $V_0 := \text{diag}(\nu_0)$, and $V_s := \text{diag}(\nu_s)$. We define $\beta_0 := X_0 V_0 e - \mu_i n e_{n_0}$ and $\beta_s := X_s V_s e - \mu_i n e_{n_s},\ s\in\mathcal{S}$. The search step is obtained by solving the linear system

$$H_0\Delta x_0 + A_0\Delta\lambda_0 + \sum_{s\in\mathcal{S}} T_s\Delta\lambda_s - \Delta\nu_0 = -\nabla_{x_0}\mathcal{L} \tag{4.4a}$$

$$H_s\Delta x_s + A_s\Delta\lambda_s - \Delta\nu_s = -\nabla_{x_s}\mathcal{L},\ s\in\mathcal{S} \tag{4.4b}$$

$$A_0^T\Delta x_0 = -\nabla_{\lambda_0}\mathcal{L} \tag{4.4c}$$

$$T_s^T\Delta x_0 + A_s^T\Delta x_s = -\nabla_{\lambda_s}\mathcal{L},\ s\in\mathcal{S} \tag{4.4d}$$

$$X_0\Delta\nu_0 + V_0\Delta x_0 = -\beta_0 \tag{4.4e}$$

$$X_s\Delta\nu_s + V_s\Delta x_s = -\beta_s,\ s\in\mathcal{S}. \tag{4.4f}$$

After eliminating the bound multipliers from the linear system we obtain

$$W_0\Delta x_0 + A_0\Delta\lambda_0 + \sum_{s\in\mathcal{S}} T_s\Delta\lambda_s = r_{x_0} \tag{4.5a}$$

$$W_s\Delta x_s + A_s\Delta\lambda_s = r_{x_s},\ s\in\mathcal{S} \tag{4.5b}$$

$$A_0^T\Delta x_0 = r_{\lambda_0} \tag{4.5c}$$

$$T_s^T\Delta x_0 + A_s^T\Delta x_s = r_{\lambda_s},\ s\in\mathcal{S}, \tag{4.5d}$$

where,

$$W_0 := H_0 + X_0^{-1}V_0 \tag{4.6a}$$

$$W_s := H_s + X_s^{-1}V_s,\ s\in\mathcal{S}. \tag{4.6b}$$

We also have that $r_{x_0} := -(\nabla_{x_0}\mathcal{L} + X_0^{-1}\beta_0)$, $r_{x_s} := -(\nabla_{x_s}\mathcal{L}_s + X_s^{-1}\beta_s)$, $r_{\lambda_0} := -\nabla_{\lambda_0}\mathcal{L}$, and $r_{\lambda_s} := -\nabla_{\lambda_s}\mathcal{L}$. The step for the bound multipliers can be recovered from

$$\Delta\nu_0 = -X_0^{-1}V_0\Delta x_0 - X_0^{-1}\beta_0 \tag{4.7a}$$

$$\Delta\nu_s = -X_s^{-1}V_s\Delta x_s - X_s^{-1}\beta_s,\ s\in\mathcal{S}. \tag{4.7b}$$

System (4.5) has the arrowhead form

$$\begin{bmatrix} K_1 & & & & B_1 \\ & K_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & K_S & B_S \\ B_1^T & B_2^T & \dots & B_S^T & K_0 \end{bmatrix} \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_S \\ \Delta w_0 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_S \\ r_0 \end{bmatrix}, \tag{4.8}$$

where $\Delta w_0^T := [\Delta x_0^T, \Delta\lambda_0^T]$, $\Delta w_s^T := [\Delta x_s^T, \Delta\lambda_s^T]$, $r_0^T := [-r_{x_0}^T, -r_{\lambda_0}^T]$, $r_s^T := [-r_{x_s}^T, -r_{\lambda_s}^T]$, and

$$K_0 := \begin{bmatrix} W_0 & A_0 \\ A_0^T & 0 \end{bmatrix}, \; K_s := \begin{bmatrix} W_s & A_s \\ A_s^T & 0 \end{bmatrix}, \; B_s := \begin{bmatrix} 0 & 0 \\ T_s^T & 0 \end{bmatrix}. \tag{4.9}$$

We refer to the linear system (4.8) as the *KKT system* and to its coefficient matrix as the *KKT matrix*. We assume that each scenario block matrix K_s, $s \in \mathcal{S}$ is nonsingular.

We use the following notation to define a block-diagonal matrix M composed of blocks $M_1, M_2, M_3, ...$:

$$M = \text{blkdiag}\{M_1, M_2, M_3, ...\}. \tag{4.10}$$

In addition, we use the following notation to define a matrix B that stacks (row-wise) the blocks $B_1, B_2, B_3...$:

$$B = \text{rowstack}\{B_1, B_2, B_3, ...\}. \tag{4.11}$$

We apply the same rowstack notation for vectors. We use the notation $v^{(k)}$ to indicate the k-th entry of vector v. We use $\text{vec}(M)$ to denote the row-column vectorization of matrix M and we define $\sigma_{min}(M)$ as the smallest singular value of matrix M. We use $\|\cdot\|$ to denote the Euclidean norm for vectors and the Frobenius norm for matrices, and we recall that $\|M\| = \|\text{vec}(M)\|$ for matrix M.

4.2 Clustering Setting

In this section we review work on scenario reduction and highlight differences and contributions of our work. We then present our clustering-based preconditioner for the KKT system (4.8).

4.2.1 Related Work and Contributions

Scenario clustering (also referred to as aggregation) is a strategy commonly used in stochastic programming to reduce computational complexity. We can classify these strategies as outside-the-solver and inside-the-solver strategies. Outside-the-solver strategies perform clustering on the scenario data (right-hand sides, matrices, and gradients) prior to the solution of the problem (de Oliveira et al., 2010; Latorre et al., 2007; Heitsch and Römisch, 2009; Casey and Sen, 2005). This approach can provide lower bounds and error bounds for linear programs (LPs) and this feature can be exploited in branch-and-bound procedures (Casey and Sen, 2005; Birge, 1985; Shetty and Taylor, 1987; Zipkin, 1980).

Outside-the-solver clustering approaches give rise to several inefficiencies, however. First, several optimization problems might need to be solved in order to refine the solution. Second, these approaches focus on the problem data and thus *do not capture the effect of the data on the particular problem at hand.* Consider, for instance, the situation in which the same scenario data (e.g., weather scenarios) is used for two very different problem classes (e.g., farm and power grid planning). Moreover, clustering scenarios based on data alone is inefficient because scenarios that are close in terms of data might have very different impact on the cost function (e.g., if they are close to the constraint boundary). Conversely, two scenarios that are distant in terms of data might have similar contributions to the cost function. We also highlight that many scenario generation procedures require knowledge of the underlying probability distributions (Dupačová et al., 2003; Heitsch and Römisch, 2009) which are often not available in closed form (e.g., weather forecasting) (Zavala et al., 2009; Lubin et al., 2011).

In this chapter, we seek to overcome these inefficiencies by performing clustering adaptively inside-the-solver. In an interior-point setting this can be done by creating a preconditioner for the KKT system (4.8) by *clustering* the scenario blocks. A key advantage of this approach is that a single optimization problem is solved and the clusters are refined only if the preconditioner is not sufficiently accurate. In addition, this approach provides a mechanism to capture the influence of the data on the particular problem at hand. Another advantage is that it can enable sparse preconditioning of Schur complement systems. This is beneficial in situations where the number of first-stage variables is large and thus Schur complement decomposition is expensive. Moreover, our approach does not require any knowledge of the underlying probability distributions generating the scenario data. Thus, it can be applied to problems in which simulators are used to generate scenarios (e.g., weather forecasting), and it can be applied to problem classes that exhibit similar structures such as support vector machines (Ferris and Munson, 2002; Jung et al., 2008) and scenario-based robust optimization (Calafiore and Campi, 2006). Our proposed clustering approach can also be used in combination with outside-the-solver scenario aggregation procedures, if desired.

Related work on inside-the-solver scenario reduction strategies includes stochastic Newton methods (Byrd et al., 2011). These approaches sample scenarios to create a smaller representation of the KKT system. Existing approaches, however, cannot handle constraints. Scenario and constraint reduction approaches for IP solvers have been presented in Chiang and Grothey (2012); Jung et al. (2012); Petra and Anitescu (2012); Colombo et al. (2011). In Jung et al. (2012), scenarios that have little influence on the step computation are eliminated from the optimality system. This influence is measured in terms of the magnitude of the constraint multipliers or in terms of the products $X_s^{-1}V_s$. In that work, it was found that a large proportion of scenarios or constraints can be eliminated without compromising convergence. The elimination potential can be limited in early iterations, however, because it is not clear which scenarios have strong or weak influence on the solution. In addition, this approach eliminates the scenarios from the problem formulation, and thus special safeguards are needed to guarantee convergence. Our proposed clustering approach does not eliminate the scenarios from the problem formulation; instead, the scenario space is compressed to construct preconditioners.

In Petra and Anitescu (2012) preconditioners for Schur systems are constructed by sampling the full scenario set. A shortcoming of this approach is that scenario outliers with strong influence might not be captured in the preconditioner. This behavior is handled more efficiently in the preconditioner proposed in Chiang and Grothey (2012) in which scenarios having strong influence on the Schur complement are retained and those that have weak influence are eliminated. A limitation of the Schur preconditioners proposed in Petra and Anitescu (2012); Chiang and Grothey (2012) is that they require a dense preconditioner for the Schur complement, which hinders scalability in problems with many first-stage variables. Our preconditioning approach enables sparse preconditioning and thus avoids forming and factorizing dense Schur complements. In addition, compared with approaches in Chiang and Grothey (2012); Jung et al. (2012); Petra and Anitescu (2012), our approach clusters scenarios instead of eliminating them (either by sampling or by measuring strong/weak influence). This enables us to handle scenario redundancies and outliers. In Colombo et al. (2011), scenarios are clustered to solve a reduced problem and the solution of this problem is used to warm-start the problem defined for the full scenario set. The approach can reduce the number of iterations of the full scenario problem; but the work per iteration is not reduced, as in our approach.

4.2.2 Clustering-Based Preconditioner

To derive our clustering-based preconditioner, we partition the full scenario set $\mathcal{S}$ into C clusters, where $C \leq S$. For each cluster $i \in \mathcal{C} := \{1..C\}$, we define a partition of the scenario set $\mathcal{S}_i \subseteq \mathcal{S}$ with $\omega_i := |\mathcal{S}_i|$ scenarios satisfying

$$\bigcup_{i \in \mathcal{C}} \mathcal{S}_i = \mathcal{S} \tag{4.12a}$$

$$\mathcal{S}_i \bigcap \mathcal{S}_j = \emptyset,\ i, j \in \mathcal{C}, j \neq i. \tag{4.12b}$$

For each cluster $i \in \mathcal{C}$, we pick an index $c_i \in \mathcal{S}_i$ to represent the cluster and we use these indexes to define the compressed set $\mathcal{R} := \{c_1, c_2, .., c_C\}$ (note that $|\mathcal{R}| = C$). We define the binary indicator $\kappa_{s,i}$, $s \in \mathcal{S}, i \in \mathcal{C}$, satisfying

$$\kappa_{s,i} = \begin{cases} 1 & \text{if } s \in \mathcal{S}_i \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

Using this notation we have that for arbitrary vectors $v_{c_i}, v_s, i \in \mathcal{C}$, the following identities hold:

$$\sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} \|v_{c_i} - v_s\| = \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{C}} \kappa_{s,i} \|v_{c_i} - v_s\| \tag{4.14a}$$

$$\sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} v_s = \sum_{s\in\mathcal{S}} v_s \tag{4.14b}$$

$$\sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} v_{c_i} = \sum_{i\in\mathcal{C}} \omega_i v_{c_i}. \tag{4.14c}$$

At this point, we have yet to define appropriate procedures for obtaining the cluster information $\mathcal{S}, \mathcal{R}, \mathcal{S}_i, \omega_i$ and $\kappa_{s,i}$. These will be discussed in Section 4.3.

Consider now the compact representation of the KKT system (4.8),

$$\underbrace{\begin{bmatrix} K_\mathcal{S} & B_\mathcal{S} \\ B_\mathcal{S}^T & K_0 \end{bmatrix}}_{:=K} \underbrace{\begin{bmatrix} q_\mathcal{S} \\ q_0 \end{bmatrix}}_{:=q} = \underbrace{\begin{bmatrix} t_\mathcal{S} \\ t_0 \end{bmatrix}}_{:=t}, \tag{4.15}$$

where

$$K_\mathcal{S} := \text{blkdiag}\{K_1, ..., K_S\} \tag{4.16a}$$

$$B_\mathcal{S} := \text{rowstack}\{B_1, ..., B_S\} \tag{4.16b}$$

$$q_\mathcal{S} := \text{rowstack}\{q_1, ..., q_S\} \tag{4.16c}$$

$$t_\mathcal{S} := \text{rowstack}\{t_1, ..., t_S\}. \tag{4.16d}$$

Here, $(t_0, t_{\mathcal{S}})$ are arbitrary right-hand side vectors and $(q_0, q_{\mathcal{S}})$ are solution vectors. If the solution vector $(q_0, q_{\mathcal{S}})$ does not exactly solve (4.15), it will induce a residual vector that we define as $\epsilon_r^T := [\epsilon_{r_0}^T, \epsilon_{r_{\mathcal{S}}}^T]$ with

$$\epsilon_{r_0} := K_0 q_0 + B_{\mathcal{S}}^T q_{\mathcal{S}} - t_0 \tag{4.17a}$$

$$\epsilon_{r_{\mathcal{S}}} := K_{\mathcal{S}} q_{\mathcal{S}} + B_{\mathcal{S}} q_0 - t_{\mathcal{S}}. \tag{4.17b}$$

The Schur system of (4.15) is given by

$$\underbrace{(K_0 - B_{\mathcal{S}}^T K_{\mathcal{S}}^{-1} B_{\mathcal{S}})}_{:=Z} q_0 = \underbrace{t_0 - B_{\mathcal{S}}^T K_{\mathcal{S}}^{-1} t_{\mathcal{S}}}_{:=t_Z}. \tag{4.18}$$

Because $K_{\mathcal{S}}$ is block-diagonal, we have that

$$Z = K_0 - \sum_{i \in \mathcal{C}} \sum_{s \in \mathcal{S}_i} B_s^T K_s^{-1} B_s \tag{4.19a}$$

$$t_Z = t_0 - \sum_{i \in \mathcal{C}} \sum_{s \in \mathcal{S}_i} B_s^T K_s^{-1} t_s. \tag{4.19b}$$

We now define the following:

$$K_{\mathcal{R}}^{\omega} := \text{blkdiag}\left\{\omega_1 K_{c_1}, \omega_2 K_{c_2}, ..., \omega_C K_{c_C}\right\} \tag{4.20a}$$

$$K_{\mathcal{R}}^{1/\omega} := \text{blkdiag}\left\{1/\omega_1 K_{c_1}, 1/\omega_2 K_{c_2}, ..., 1/\omega_C K_{c_C}\right\} \tag{4.20b}$$

$$B_{\mathcal{R}} := \text{rowstack}\left\{B_{c_1}, B_{c_2}, ..., B_{c_C}\right\} \tag{4.20c}$$

$$t_{\mathcal{R}} := \text{rowstack}\left\{t_{c_1}, t_{c_2}, ..., t_{c_C}\right\}. \tag{4.20d}$$

In other words, $K_{\mathcal{R}}^{\omega}$ is a block-diagonal matrix in which each block entry K_{c_i} is weighted by the scalar weight ω_i and $K_{\mathcal{R}}^{1/\omega}$ is a block-diagonal matrix in which each block entry K_{c_i} is weighted by $1/\omega_i$. Note that

$$(K_{\mathcal{R}}^{1/\omega})^{-1} = (K_{\mathcal{R}}^{-1})^{\omega}, \tag{4.21}$$

where,

$$(K_{\mathcal{R}}^{-1})^{\omega} := \text{blkdiag}\left\{\omega_1 K_{c_1}^{-1}, \omega_2 K_{c_2}^{-1}, ..., \omega_C K_{c_C}^{-1}\right\}. \tag{4.22}$$

We now present the *clustering-based preconditioner* (CP),

$$\begin{bmatrix} K_{\mathcal{R}}^{1/\omega} & B_{\mathcal{R}} \\ B_{\mathcal{R}}^T & K_0 \end{bmatrix} \begin{bmatrix} \cdot \\ q_0 \end{bmatrix} = \begin{bmatrix} t_{\mathcal{R}} \\ t_0 + t_{CP} \end{bmatrix} \tag{4.23a}$$

$$K_s q_s = t_s - B_s q_0, \quad i \in \mathcal{C},\ s \in \mathcal{S}_i, \tag{4.23b}$$

where

$$t_{CP} := \sum_{i\in\mathcal{C}} \omega_i B_{c_i}^T K_{c_i}^{-1} t_{c_i} - \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B_s^T K_s^{-1} t_s \tag{4.24}$$

is a correction term that is used to establish consistency between CP and the KKT system. In particular, the Schur system of (4.23a) is

$$\begin{aligned} \bar{Z} q_0 &= t_0 + t_{CP} - B_{\mathcal{R}}^T (K_{\mathcal{R}}^{1/\omega})^{-1} t_{\mathcal{R}} \\ &= t_0 + t_{CP} - \sum_{i\in\mathcal{C}} \omega_i B_{c_i}^T K_{c_i}^{-1} t_{c_i} \\ &= t_0 - \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B_s^T K_s^{-1} t_s \\ &= t_Z, \end{aligned} \tag{4.25}$$

with

$$\begin{aligned} \bar{Z} &:= K_0 - \sum_{i\in\mathcal{C}} \omega_i B_{c_i}^T K_{c_i}^{-1} B_{c_i} \\ &= K_0 - \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B_{c_i}^T K_{c_i}^{-1} B_{c_i}. \end{aligned} \tag{4.26}$$

Consequently, the Schur system of the preconditioner and of the KKT system have the same right-hand side. This property is key to establishing spectral and error properties for the preconditioner. In particular, note that the solution of CP system (4.23a)-(4.23b) solves the perturbed KKT system,

$$\underbrace{\begin{bmatrix} K_{\mathcal{S}} & B_{\mathcal{S}} \\ B_{\mathcal{S}}^T & K_0 + E_Z \end{bmatrix}}_{:=\bar{K}} \begin{bmatrix} q_{\mathcal{S}} \\ q_0 \end{bmatrix} = \begin{bmatrix} t_{\mathcal{S}} \\ t_0 \end{bmatrix}, \tag{4.27}$$

where

$$E_Z := \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B_s^T K_s^{-1} B_s - \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B_{c_i}^T K_{c_i}^{-1} B_{c_i}, \tag{4.28}$$

is the Schur error matrix and satisfies $Z + E_Z = \bar{Z}$. The mathematical equivalence between CP system (4.23a)-(4.23b) and (4.27) can be established by constructing the Schur system of (4.27) and noticing that it is equivalent to (4.25). Moreover, the second-stage steps are the same. Consequently, *applying preconditioner CP is equivalent to using the perturbed matrix $\bar{K}$ as a preconditioning matrix for the KKT matrix K.* We will use this equivalence to establish spectral and error properties in Section 4.3.

The main idea behind preconditioner CP (we will use CP for short) is to compress the KKT system (4.15) into the smaller system (4.23a) which is cheaper to factorize. We solve this smaller system to obtain q_0, and we recover $q_{\mathcal{S}}$ from (4.23b) by factorizing the individual blocks K_s. We refer to the coefficient matrix of (4.23a) as the *compressed matrix.*

In the following, we assume that the Schur complements Z and $\bar{Z}$ are nonsingular. The nonsingularity of Z together with the assumption that all the blocks K_s are nonsingular implies (from the Schur complement theorem) that matrix K defined in (4.15) is nonsingular and thus the KKT system has a unique solution. The nonsingularity of $\bar{Z}$ together with the assumption that all the blocks K_s are nonsingular implies that the compressed matrix is nonsingular and thus CP has a unique solution. Note that we could have also assumed nonsingularity of matrix K directly and this, together with the nonsingularity of the blocks K_s, would imply nonsingularity of Z (this also from the Schur complement theorem). The same applies if we assume nonsingularity of the compressed matrix, which would imply nonsingularity of $\bar{Z}$.

Although Schur complement decomposition is a popular approach for solving structured KKT systems, it suffers from poor scalability with the dimension of q_0. The reason is that the Schur complement needs to be formed (this requires as many backsolves with the factors of K_s as the dimension of q_0) and factored (this requires a factorization of a dense matrix of dimension q_0). We elaborate on these scalability issues in Section 4.4. We thus highlight that the Schur system representations are used only for analyzing CP.

Our preconditioning setting is summarized as follows. At each IP iteration k, we compute a step by solving the KKT system (4.8). We do so by finding a solution vector $(\Delta w_0, \Delta w_{\mathcal{S}})$ of the ordered KKT system (4.8) for the right-hand side $(r_0, r_{\mathcal{S}})$ using an iterative linear algebra solver such as GMRES, QMR, or BICGSTAB. Here, $(r_0, r_{\mathcal{S}})$ are the right-hand side vectors of the KKT system (4.8) in ordered form. Each minor iteration of the iterative linear algebra solver is denoted by $\ell = 0, 1, 2, ..,$. We denote the initial guess of the solution vector of (4.8) as $(\Delta w_0^\ell, \Delta w_{\mathcal{S}}^\ell)$ with $\ell = 0$. At each minor iterate ℓ, the iterative solver will request the application of CP to a given vector $(t_0^\ell, t_{\mathcal{S}}^\ell)$, and the solution vectors $(q_0^\ell, q_{\mathcal{S}}^\ell)$ of (4.23) are returned to the iterative linear algebra solver. *Perfect preconditioning* occurs when we solve (4.8) instead of (4.23) with the right-hand sides $(t_0^\ell, t_{\mathcal{S}}^\ell)$.

4.3 Preconditioner Properties

In this section we establish properties for CP and we use these to guide the design of appropriate clustering strategies. The relationship between the CP system (4.23) and the perturbed KKT system (4.27) allows us to establish the following result.

Lemma 1 *The preconditioned matrix $\bar{K}^{-1}K$ has $(n+m-n_0-m_0)$ unit eigenvalues, and the remaining (n_0+m_0) eigenvalues are bounded as*

$$|\lambda(\bar{K}^{-1}K) - 1| \leq \frac{1}{\sigma_{min}(\bar{Z})}\|E_Z\|.$$

Proof: The eigenvalues λ and eigenvectors $w := (w_{\mathcal{S}}, w_0)$ of $\bar{K}^{-1}K$ satisfy $\bar{K}^{-1}Kw = \lambda w$, and thus $Kw = \lambda \bar{K} w$. Consequently,

$$\begin{aligned} K_{\mathcal{S}} w_{\mathcal{S}} + B_{\mathcal{S}} w_0 &= \lambda(K_{\mathcal{S}} w_{\mathcal{S}} + B_{\mathcal{S}} w_0) \\ B_{\mathcal{S}}^T w_{\mathcal{S}} + K_0 w_0 &= \lambda B_{\mathcal{S}}^T w_{\mathcal{S}} + \lambda(K_0 + E_Z) w_0. \end{aligned}$$

From the first relationship we have $n+m-n_0-m_0$ unit eigenvalues. Applying Schur complement decomposition to the eigenvalue system, we obtain

$$\begin{aligned} Z w_0 &= \lambda(Z + E_Z) w_0 \\ &= \lambda \bar{Z} w_0. \end{aligned}$$

We can thus express the remaining $n_0 + m_0$ eigenvalues of $\bar{K}^{-1}K$ as $\lambda = 1 + \epsilon_Z$ to obtain

$$\begin{aligned} |\epsilon_Z| &= \frac{\|E_Z w_0\|}{\|\bar{Z} w_0\|} \\ &\leq \frac{1}{\sigma_{min}(\bar{Z})}\|E_Z\|. \end{aligned}$$

The proof is complete. □

The above lemma is a direct consequence of Theorem 3.1 in Dollar (2007). From the definition of E_Z we note that the following bound holds:

$$|\lambda(\bar{K}^{-1}K) - 1| \leq \frac{1}{\sigma_{min}(\bar{Z})} \sum_{i \in \mathcal{C}} \sum_{s \in \mathcal{S}_i} \left\| B_s^T K_s^{-1} B_s - B_{c_i}^T K_{c_i}^{-1} B_{c_i} \right\|. \tag{4.30}$$

Lemma 1 states that we can improve the spectrum of $\bar{K}^{-1}K$ by choosing clusters that minimize $\|E_Z\|$. This approach, however, would require expensive matrix operations. An interesting and tractable exception occurs when $Q_s = Q$, $W_s = W$, and $T_s = T,\ i \in \mathcal{C}, s \in \mathcal{S}_i$. This case is quite common in applications and arises when the scenario data is only defined by the right-hand sides b_s and the cost coefficients d_s of (4.1). We refer to this case as the *special data case.* In this case we have that E_Z reduces to

$$E_Z = \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B^T \left(K_s^{-1} - K_{c_i}^{-1}\right) B. \tag{4.31}$$

We also have that K_s and K_{c_i} differ only in the diagonal matrices $X_s^{-1}V_s$ and $X_{c_i}^{-1}V_{c_i}$. We thus have,

$$K_s - K_{c_i} = \begin{bmatrix} (X_s^{-1}V_s - X_{c_i}^{-1}V_{c_i}) & 0 \\ 0 & 0 \end{bmatrix}. \tag{4.32}$$

If we define the vectors,

$$\gamma_s = \text{vec}(X_s^{-1}V_s), i \in \mathcal{C}, s \in \mathcal{S}_i \tag{4.33a}$$

$$\gamma_{c_i} = \text{vec}(X_{c_i}^{-1}V_{c_i}), i \in \mathcal{C}, \tag{4.33b}$$

we can establish the following result.

Theorem 4.3.1 *Assume that $Q_s = Q$, $W_s = W$, and $T_s = T,\ i \in \mathcal{C}, s \in \mathcal{S}_i$ holds. Let vectors γ_s, γ_{c_i} be defined as in (4.33). The preconditioned matrix $\bar{K}^{-1}K$ has $(n + m - n_0 - m_0)$ unit eigenvalues, and there exists a constant $c_K > 0$ such that the remaining $(n_0 + m_0)$ eigenvalues are bounded as*

$$|\lambda(\bar{K}^{-1}K) - 1| \leq \frac{c_K}{\sigma_{min}(\bar{Z})} \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{C}} \kappa_{s,i} \|\gamma_{c_i} - \gamma_s\|.$$

Proof: From Lemma 1 we have that $n_0 + m_0$ eigenvalues λ of $\bar{K}^{-1}K$ are bounded as $|\lambda - 1| \leq \frac{1}{\sigma_{min}(\bar{Z})}\|E_Z\|$. We define the error matrix,

$$E_s := K_s - K_{c_i},\ i \in \mathcal{C}, s \in \mathcal{S}_i$$

and use (4.31) and (4.32) to obtain the bound,

$$\begin{aligned}\|E_Z\| &\leq \sum_{i\in\mathcal{C}}\sum_{s\in\mathcal{S}_i}\|B^TB\|\|K_s^{-1} - K_{c_i}^{-1}\| \\ &= \sum_{i\in\mathcal{C}}\sum_{s\in\mathcal{S}_i}\|B^TB\|\|(K_{c_i}+E_s)^{-1} - K_{c_i}^{-1}\|.\end{aligned}$$

We have that

$$\begin{aligned}(K_{c_i}+E_s)^{-1} - K_{c_i}^{-1} &= -(K_{c_i}+E_s)^{-1}E_sK_{c_i}^{-1} \\ &= -K_s^{-1}E_sK_{c_i}^{-1}.\end{aligned}$$

This can be verified by multiplying both sides by $K_{c_i} + E_s$. We thus have

$$\begin{aligned}\|E_Z\| &\leq \sum_{i\in\mathcal{C}}\sum_{s\in\mathcal{S}_i}\|B^TB\|\|(K_{c_i}+E_s)^{-1} - K_{c_i}^{-1}\| \\ &\leq \sum_{i\in\mathcal{C}}\sum_{s\in\mathcal{S}_i}\|B^TB\|\|K_s^{-1}\|\|K_{c_i}^{-1}\|\|E_s\| \\ &\leq c_K\sum_{i\in\mathcal{C}}\sum_{s\in\mathcal{S}_i}\|\text{vec}(X_{c_i}^{-1}V_{c_i}) - \text{vec}(X_s^{-1}V_s)\|,\end{aligned}$$

with $c_K := \sum_{i\in\mathcal{C}}\sum_{s\in\mathcal{S}_i}\|B^TB\|\|K_s^{-1}\|\|K_{c_i}^{-1}\|$. The existence of c_K follows from the nonsingularity of K_s and K_{c_i}. The proof is complete. $\square$

We now develop a bound of the preconditioning error for the *general data case* in which the scenario data is also defined by coefficient matrices. Notably, this bound does not require the minimization of the error $\|E_Z\|$. The idea is to bound the error induced by CP relative to the exact solution of the KKT system (4.15) (perfect preconditioner). This approach is used to characterize inexact preconditioners such as multigrid and nested preconditioned conjugate gradient (Szyld and Vogel, 2001). We express the solution of CP obtained from (4.23) as $q^T = [q_{\mathcal{S}}^T, q_0^T]$ and that of the KKT system (4.15) as $q^{*T} = [q_{\mathcal{S}}^{*T}, q_0^{*T}]$. We define the error between q and q^* as $\epsilon := q - q^*$ and we seek to bound ϵ. If we decompose the error as $\epsilon^T = [\epsilon_{\mathcal{S}}^T, \epsilon_0^T]$, we have that $\epsilon_0 = q_0 - q_0^*$ and $\epsilon_{\mathcal{S}} = q_{\mathcal{S}} - q_{\mathcal{S}}^*$.

We recall that the Schur systems of (4.15) and of (4.23) and their respective solutions satisfy

$$Zq_0^* = t_Z \tag{4.34a}$$

$$\bar{Z}q_0 = t_Z. \tag{4.34b}$$

If we define the vectors,

$$\gamma_s = (B_s^T K_s^{-1} B_s)t_Z, i \in \mathcal{C}, s \in \mathcal{S}_i \tag{4.35a}$$

$$\gamma_{c_i} = (B_{c_i}^T K_{c_i}^{-1} B_{c_i})t_Z, i \in \mathcal{C}. \tag{4.35b}$$

we can establish the following bound on the error $\epsilon = q - q^*$.

Lemma 2 *Assume that there exists $c_T > 0$ such that $\|(Z - \bar{Z})Z^{-1}t_Z\| \leq c_T\|(Z - \bar{Z})t_Z\|$ holds; then there exists $c_{ZK} > 0$ such that the preconditioner error ϵ is bounded as*

$$\|\epsilon\| \leq c_{ZK}\|(Z - \bar{Z})t_Z\|.$$

Proof: From $\epsilon_0 = q_0 - q_0^*$ we have $\bar{Z}\epsilon_0 = \bar{Z}q_0 - \bar{Z}q_0^*$. From (4.34) we have $\bar{Z}q_0 = Zq_0^* = t_Z$ and thus $\bar{Z}\epsilon_0 = Zq_0^* - \bar{Z}q_0^*$. We thus have,

$$\begin{aligned}
\bar{Z}\epsilon_0 &= Zq_0^* - \bar{Z}q_0^* \\
&= t_Z - \bar{Z}q_0^* \\
&= t_Z - \bar{Z}Z^{-1}t_Z \\
&= t_Z - (Z + (\bar{Z} - Z))Z^{-1}t_Z \\
&= (Z - \bar{Z})Z^{-1}t_Z
\end{aligned}$$

We recall that

$$\begin{aligned}
q_{\mathcal{S}}^* &= K_{\mathcal{S}}^{-1}(t_{\mathcal{S}} - B_{\mathcal{S}}q_0^*) \\
q_{\mathcal{S}} &= K_{\mathcal{S}}^{-1}(t_{\mathcal{S}} - B_{\mathcal{S}}q_0)
\end{aligned}$$

and thus

$$\begin{aligned}
\epsilon_{\mathcal{S}} &= K_{\mathcal{S}}^{-1}B_{\mathcal{S}}(q_0^* - q_0) \\
&= -K_{\mathcal{S}}^{-1}B_{\mathcal{S}}\epsilon_0.
\end{aligned}$$

We thus have

$$\begin{aligned}
\|\epsilon_0\| &\leq c_Z\|(Z - \bar{Z})t_Z\| \\
\|\epsilon_\Omega\| &\leq c_{K_{\mathcal{S}}}\|\epsilon_0\|,
\end{aligned}$$

with $c_Z := \|\bar{Z}^{-1}\|c_T$ and $c_{K_{\mathcal{S}}} := \|K_{\mathcal{S}}^{-1}B_{\mathcal{S}}\|$. The existence of c_Z follows from the assumption that $\bar{Z}$ is nonsingular. The existence of $c_{\mathcal{S}}$ follows from the assumption that the blocks K_s are nonsingular and thus $K_{\mathcal{S}}$ is nonsingular. The result follows from $\|\epsilon\| \leq \|\epsilon_0\| + \|\epsilon_{\mathcal{S}}\|$ and by defining $c_{ZK} := c_Z(1 + c_{K_{\mathcal{S}}})$. $\square$

The assumption that there exists $c_T > 0$ such that $\|(Z - \bar{Z})Z^{-1}t_Z\| \leq c_T\|(Z - \bar{Z})t_Z\|$ holds is trivially satisfied when Z^{-1} and $\bar{Z}$ commute (i.e., $\bar{Z}Z^{-1}$ is a symmetric matrix). In this case we have that $c_T = \|Z^{-1}\|$. The matrices also commute in the limit $\bar{Z} \to Z$ because $\bar{Z}Z^{-1} = ZZ^{-1} + (\bar{Z} - Z)Z^{-1}$ and thus $\bar{Z}Z^{-1} \to I$. When Z and $\bar{Z}$ do not commute we require that $\|(Z - \bar{Z})Z^{-1}t_Z\|$ decreases when $\|(Z - \bar{Z})t_Z\|$ does. We validate this empirically in Section 4.4.

Theorem 4.3.2 *Let vectors γ_s, γ_{c_i} be defined as in* (4.35)*. The preconditioner error ϵ is bounded as*

$$\|\epsilon\| \leq c_{ZK} \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{C}} k_{s,i} \|\gamma_{c_i} - \gamma_s\|,$$

with c_{ZK} defined in Lemma 2.

Proof: From (4.35) and (4.28) we have that

$$\begin{aligned}
\bar{Z}t_Z - Zt_Z &= E_Z t_Z \\
&= \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B_s^T K_s^{-1} B_s t_Z - \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B_{c_i}^T K_{c_i}^{-1} B_{c_i} t_Z \\
&= \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} (B_s^T K_s^{-1} B_s t_Z - B_{c_i}^T K_{c_i}^{-1} B_{c_i} t_Z) \\
&= \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} (\gamma_s - \gamma_{c_i}).
\end{aligned}$$

We bound this expression to obtain,

$$\begin{aligned}
\|\bar{Z}t_Z - Zt_Z\| &= \left\| \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} (\gamma_s - \gamma_{c_i}) \right\| \\
&\leq \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} \|\gamma_{c_i} - \gamma_s\| \\
&= \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{C}} \kappa_{s,i} \|\gamma_{c_i} - \gamma_s\|.
\end{aligned}$$

The result follows from Lemma 2. □

We can see that the properties of CP are related to a metric of the form

$$\mathcal{D}_C := \sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{C}} \kappa_{s,i}\|\gamma_{c_i} - \gamma_s\|. \tag{4.36}$$

This is the *distortion metric* widely used in clustering analysis (Bishop et al., 2006). The distortion metric is (partially) minimized by K-means, K-medoids, and hierarchical clustering algorithms to determine $\kappa_{s,i}$ and γ_{c_i}. The vectors γ_s are called *features*, and γ_{c_i} is the centroid of cluster $i \in \mathcal{C}$ (we can also pick the scenario that is closest to the centroid if the centroid is not an element of the scenario set). The distortion metric is interpreted as the accumulated distance of the elements of the cluster relative to the centroid. If the distortion is small, then the scenarios in a cluster are similar. The distortion metric can be made arbitrarily small by increasing the number of clusters and is zero in the limit with $S = C$ because each cluster is given by one scenario. Consequently, we see that Theorems 4.3.1 and 4.3.2 provide the necessary insights to derive clusters using different sources of information of the scenarios.

Theorem 4.3.1 suggests that, in the special data case with features defined as $\gamma_s = \text{vec}(X_s^{-1}V_s)$, the spectrum of $\bar{K}^{-1}K$ can be made arbitrarily close to one if the distortion metric is made arbitrarily small. This implies that the *definition of the features is consistent.* We highlight, however, that the bounds of Theorem 4.3.1 assume that the clustering parameters are given (i.e., the sets $\mathcal{C}$ and $\mathcal{C}_i$ are fixed). Consequently, the constants c_K, and $\sigma_{min}(\bar{Z})$ change when the clusters are changed. Because of this, we cannot guarantee that reducing the distortion metric will indeed improve the quality of the preconditioner. The aforementioned constants depend in nontrivial ways on the clustering parameters and it is thus difficult to obtain bounds for them. In the next section we demonstrate empirically, however, that the constants c_K and $\sigma_{min}(\bar{Z})$ are insensitive to the clustering parameters. Consequently, reducing the distortion metric in fact improves the quality of the preconditioner. We leave the theoretical treatment of this issue as part of future work.

We can obtain useful insights from the special data case. First note that the scenarios are clustered at each IP iteration k because the matrices $X_s^{-1}V_s$ change along the search. The clustering approach is therefore adaptive, unlike outside-the-solver scenario clustering approaches. In fact, it is not possible to derive spectral and error properties for preconditioners based on clustering of problem data alone. Our approach focuses directly on the contributions $X_s^{-1}V_s$ and thus assumes that the problem data enters indirectly through the contributions $X_s^{-1}V_s$, which in turn affect the structural properties of the KKT matrix. The features $\gamma_s = \text{vec}(X_s^{-1}V_s)$ have an important interpretation: these reflect the contribution of each scenario to the logarithmic barrier function. From complementarity we have that $\|X_s\| \gg 0$ implies $\|V_s\| \approx 0$ and $\|X_s^{-1}V_s\| \approx 0$. In this case we say that there is weak activity in the scenario and we have from (4.6) that $W_s = H_s + X_s^{-1}V_s \approx H_s$. Consequently, the primal-dual term $X_s^{-1}V_s$ for a scenario with weak activity puts little weight on the barrier function. In the opposite case in which the scenario has strong activity we have that $\|V_s\| \gg 0$, $\|X_s\| \approx 0$, and $\|X_s^{-1}V_s\| \gg 0$. In this case we thus have that a scenario with strong activity puts a large weight on the barrier function. This reasoning is used in Jung et al. (2012); Gondzio and Grothey (2003) to eliminate the scenarios with weak activity. In our case we propose to cluster scenarios with similar activities. Clustering allows us to eliminate redundancies in both active and inactive scenarios and to capture outliers. In addition, this strategy avoids the need to specify a threshold to classify weak and strong activity.

Theorem 4.3.2 provides a mechanism to obtain clusters for the general data case in which the scenario data is defined also by the coefficient matrices. The result states that we can bound the preconditioning error using the Schur complement error $E_Z = \bar{Z} - Z$ projected on the right-hand side vector t_Z. Consequently, the error can be bounded by the distortion metric with features defined in (4.35). This suggests that the error can be made arbitrarily small if the distortion is made arbitrarily small.

Moreover, it is not necessary to perform major matrix operations. As in the special data case of Theorem 4.3.1, however, the bounding constant c_Z of Theorem 4.3.2 depends on the clustering parameters. Moreover, we need to verify that the term $\|(Z - \bar{Z})Z^{-1}t_Z\|$ decreases when $\|(Z - \bar{Z})t_Z\|$ does. In the next section we verify these two assumptions empirically.

The error bound of Theorem 4.3.2 requires that clustering tasks and the factorization of the compressed matrix be performed at each minor iteration ℓ of the iterative linear algebra solver. The reason is that the features (4.35) change with t_Z^ℓ. Performing these tasks at each minor iteration, however, is expensive. Consequently, we perform these tasks only at the first minor iteration $\ell = 0$. If the initial guess of the solution vector of the KKT system is set to zero ($\Delta w_0^\ell = 0$ and $\Delta w_{\mathcal{S}}^\ell = 0$) and if GMRES, QMR, or BICGSTAB schemes are used, this is equivalent to performing by clustering using the features

$$\gamma_s = (B_s^T K_s^{-1} B_s) r_Z, i \in \mathcal{C}, s \in \mathcal{S}_i \tag{4.37a}$$

$$\gamma_{c_i} = (B_{c_i}^T K_{c_i}^{-1} B_{c_i}) r_Z, i \in \mathcal{C}, \tag{4.37b}$$

where

$$\begin{aligned} r_Z &= t_Z^0 \\ &= r_0 - \sum_{i \in \mathcal{C}} \sum_{s \in \mathcal{S}_i} B_s^T K_s^{-1} r_s \end{aligned} \tag{4.38}$$

is the right-hand side of the Schur system of (4.8).

4.4 Numerical Results

In this section we discuss implementation issues of CP and present numerical results for benchmark problems in the literature and a large-scale stochastic market clearing problem. We begin by summarizing the procedure for computing the step

$(\Delta x^k, \Delta\lambda^k, \Delta\nu^k)$ at each IP iteration k.

Step Computation Scheme

1. **Initialization**. Given iterate (x^k, λ^k, ν^k), number of clusters C, tolerance τ^k, and maximum number of linear solver iterates m_{it}.

2. **Get Clustering Information.**

   2.0. Compute features $\gamma_s,\ s \in \mathcal{S}$ as in (4.33) or (4.37).

   2.1. Obtain $\kappa_{s,i}$ and γ_{c_i} using K-means, hierarchical clustering, or any other clustering algorithm.

   2.2. Use $\kappa_{s,i}$ to construct $\mathcal{C}$, $\mathcal{R}$, Ω, and ω_i.

   2.3. Construct and factorize compressed matrix

$$\begin{bmatrix} K_{\mathcal{R}}^{1/\omega} & B_{\mathcal{R}} \\ B_{\mathcal{R}}^T & K_0 \end{bmatrix}$$

   and factorize scenario matrices $K_s,\ i \in \mathcal{C}, s \in \mathcal{S}_i$.

3. **Get Step.**

   3.1. Call iterative linear solver to solve KKT system (4.15) with right-hand sides $(r_0, r_{\mathcal{S}})$, set $\ell = 0$, and initial guess $\Delta w_0^\ell = 0$ and $\Delta w_{\mathcal{S}}^\ell = 0$. At each minor iterate $\ell = 0, 1, ...,$ of the iterative linear solver, DO:

   3.1.1. Use factorization of compressed matrix and of $K_{\mathcal{S}}$ to solve CP (4.23a)-(4.23b) for right-hand sides $(t_0^\ell,\, t_{\mathcal{S}}^\ell)$ and RETURN solution $(q_0^\ell, q_{\mathcal{S}}^\ell)$.

   3.1.2. From (4.17), get ϵ_r^ℓ using solution vector $(\Delta w_0^\ell, \Delta w_{\mathcal{S}}^\ell)$ and right-hand side vectors $(r_0, r_{\mathcal{S}})$. If $\|\epsilon_r^\ell\| \leq \tau^k$, TERMINATE.

   3.1.3. If $\ell = m_{it}$, increase C, and RETURN to Step 3.1.

   3.2. Recover $(\Delta x^k, \Delta\lambda^k)$ from $(\Delta w_0^\ell, \Delta w_{\mathcal{S}}^\ell)$.

3.3. Recover $\Delta\nu^k$ from (4.7).

We call our clustering-based IP framework `IP-CLUSTER`. The framework is written in C++ and uses MPI for parallel computations. In this implementation we use the primal-dual IP algorithm of Mehrotra (Mehrotra, 1992). We use the matrix templates and direct linear algebra routines of the `BLOCK-TOOLS` library (Kang et al., 2014). This library is specialized to block matrices and greatly facilitated the implementation. Within `BLOCK-TOOLS`, we use its MA57 interface to perform all direct linear algebra operations. We use the `GMRES` implementation within the `PETSc` library (`http://www.mcs.anl.gov/petsc`) to perform all iterative linear algebra operations. We have implemented serial and parallel versions of CP. We highlight that the parallel version performs the factorizations of (4.23b) in parallel and exploits the block-bordered-diagonal structure of the KKT matrix to perform matrix-vector operations in parallel as well. We use the K-means and hierarchical clustering implementations of the `C-Clustering` library (`http://bonsai.hgc.jp/~mdehoon/software/cluster/software.htm`). To implement the market clearing models we use an interface to `AMPL` to create individual instances (.nl files) for each scenario and indicate first-stage variables and constraints using the `suffix` capability.

4.4.1 Benchmark Problems

We consider stochastic variants of problems obtained from the CUTEr library and benchmark problems (SSN, GBD, LANDS, 20TERM) reported in Linderoth et al. (2006). The deterministic CUTEr QP problems have the form

$$\min \frac{1}{2}x^T Hx + d^T x, \text{ s.t. } Ax = b, \; x \geq 0. \tag{4.39}$$

We generate a stochastic version of this problem by defining b as a random vector. We create scenarios for this vector $b_s, s \in \mathcal{S}$ using the nominal value b as mean and a standard deviation $\pm\sigma = 0.5b$. We then formulate the two-stage stochastic program:

$$\min\ e^T x_0 + \sum_{s\in\mathcal{S}} \xi_s \left(\frac{1}{2} x_s^T H x_s + d^T x_s\right) \tag{4.40a}$$

$$\text{s.t.} \quad Ax_s = b_s, s \in \mathcal{S} \tag{4.40b}$$

$$x_s + x_0 \geq 0,\ s \in \mathcal{S} \tag{4.40c}$$

$$x_0 \geq 0. \tag{4.40d}$$

Here, we set $\xi_s = 1/|\mathcal{S}|$. We first demonstrate the quality of CP in terms of the number of GMRES iterations. For all cases, we assume a scenario compression rate of 75% (only 25% of the scenarios are used in the compressed matrix), and we solve the problems to a tolerance of 1×10^{-6}. We use the notation x% to indicate the compression rate (i.e., the preconditioner CP uses 100-x% of the scenarios to define the compressed matrix). A compression rate of 0% indicates that the entire scenario set is used for the preconditioner (ideal). A compression rate of 100% indicates that no preconditioner is used. We set the maximum number of GMRES iterations m_{it} to 100.

For this first set of results we cluster the scenarios using a hierarchical clustering algorithm with the features (4.35). The results are presented in Table 4.1. As can be seen, the performance of CP is satisfactory in all instances, requiring fewer than 20 GMRES iterations per interior-point iteration (this is labeled as LAit/IPit). We attribute this to the particular structure of CP, which enable us to pose the preconditioning systems in the equivalent form (4.27) and to derive favorable spectral properties and error bounds. To support these observations, we have also experi-

Table 4.1: Performance of naive and preconditioner CPs in benchmark problems.

				NPI (75%)			CP (75%)			NPII (75%)		
Problem	S	n	m	IPit	LAit	LAit/IPit	IPit	LAit	LAit/IPit	IPit	LAit	LAit/IPit
HS53	100	1,010	800	19	152	8	19	113	5	20	112	5
LOTSCHD	100	1,212	700	27	911*	33	25	203	8	26	178	6
HS76	100	707	300	24	626*	26	23	98	4	24	107	4
HS118	100	5,959	4,400	47	1499*	31	47	409	8	50	484	9
QPCBLEND	100	11,514	7,400	57	258	4	57	253	4	55	273	4
ZECEVIC2	100	606	400	27	451*	16	29	111	3	26	107	4
QPTEST	100	505	300	23	569*	24	23	108	4	26	109	4
SSN	100	70,689	8,600	114	738*	6	114	1857	16	114	2506	21
GBD	1000	10,017	5,000	24	627*	26	24	144	6	24	92	4
LANDS	1000	12,004	7,003	29	481*	16	29	115	3	29	122	4
20TERM	100	76,463	12,404	57	581*	10	57	976	17	58	905*	16

mented with a couple of *naive* preconditioners. The first naive preconditioner (NPI) has the form:

$$\begin{bmatrix} \bar{K}_{\mathcal{S}} & \bar{B}_{\mathcal{S}} \\ \bar{B}_{\mathcal{S}}^T & K_0 \end{bmatrix} \begin{bmatrix} q_{\mathcal{S}} \\ q_0 \end{bmatrix} = \begin{bmatrix} t_{\mathcal{S}} \\ t_0 \end{bmatrix}, \tag{4.41}$$

where,

$$\bar{K}_{\mathcal{S}} := \text{blkdiag}\left\{K_{c_1}, ..., K_{c_1}, K_{c_2}, ..., K_{c_2}, ..., K_{c_C}, ..., K_{c_C}\right\} \tag{4.42a}$$

$$\bar{B}_{\mathcal{S}} := \text{rowstack}\left\{B_{c_1}, ..., B_{c_1}, B_{c_2}, ..., B_{c_2}, ..., B_{c_C}, ..., B_{c_C}\right\}. \tag{4.42b}$$

We can see that NPI replaces the block matrix elements of the cluster with those of the scenario representing the cluster. This, in fact, is done implicitly by the compressed system (4.23a). Note also that the right-hand side of NPI is consistent with that of the KKT system. The design of NPI seems reasonable at first sight but it has several structural deficiencies. We highlight these by noticing that the Schur system of NPI has the form

$$\bar{Z}q_0 = t_0 - \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}_i} B_{c_i}^T K_{c_i}^{-1} t_s. \tag{4.43}$$

This system has the same Schur matrix as that of the Schur system of CP (4.25) but does not have the same right-hand side. Moreover, the second-stage steps obtained from NPI are

$$K_{c_i} q_s = t_s - B_{c_i} q_0, \quad i \in \mathcal{C},\ s \in \mathcal{S}_i. \tag{4.44}$$

By comparing (4.44) with (4.23b) we can see that the recovery of the second-stage steps in NPI does not use the second-stage matrices K_s, B_s corresponding to each scenario (as is done in CP approach). We propose an alternative naive preconditioner (NPII) to analyze the impact of the second-stage step (4.23b). This preconditioner computes q_0 using (4.41) as in NPI but computes the second-stage steps using (4.44) as in CP. Consequently, NPII and CP only differ in the way q_0 is computed. It is not difficult to verify that the solution of NPII is equivalent to the solution of the system:

$$\begin{bmatrix} K_{\mathcal{S}} & B_{\mathcal{S}} \\ B_{\mathcal{S}}^T & K_0 + E_Z \end{bmatrix} \begin{bmatrix} q_{\mathcal{S}} \\ q_0 \end{bmatrix} = \begin{bmatrix} t_{\mathcal{S}} \\ t_0 + t_{CP} \end{bmatrix}. \tag{4.45}$$

By comparing the equivalent system (4.27) of CP and the equivalent system (4.45) of NPII we can see that NPII introduces the additional perturbation t_{CP} on the right-hand side.

The structural deficiencies of NPI and NPII prevent us from obtaining the error bounds of Theorems 4.3.1 and 4.3.2 and highlight the importance of the structure of CP. In Table 4.1 we compare the performance of the different preconditioners. We can see that the performance of NPI is not competitive. In particular, CP outperforms NPI in nine instances out of eleven. Moreover, in all instances except HS53 and

QPCBLEND, it was necessary to refine preconditioner NPI in several iterations (this is done by increasing the number of clusters). We highlight instances in which this occurs using a star next to the total number of GMRES iterations. The performance of NPII is highly competitive with that of CP. In fact, NPII performs slightly better than CP in several instances. For problem 20TERM, however, it was necessary to increase the number of clusters for NPII in some iterations. We can thus conclude that CP has in general better performance and is more robust. Moreover, we can conclude that the second-stage step (4.23b) plays a key role.

In Table 4.2 we compare the performance of CP with that of the unpreconditioned strategy (compression rate of 100%) and with that of the naive strategies. We only show results for a single instance to illustrate that the matrices of the benchmark problems are nontrivial and preconditioning is indeed needed.

Table 4.2: Performance of preconditioned and unpreconditioned strategies.

Problem	S	n	Compress.	IPit	LAit	LAit/IPit
HS53	100	1,010	100%	19	12861	676
			75% (NPI)	19	152	8
			75% (CP)	19	113	5
			75% (NPII)	20	112	5

We note that the instances reported in Tables 4.1 and 4.2 are small ($n < 100,000$). In most of these small instances we found that the solution times obtained with full factorization are shorter than those obtained with CP. This is because the overhead introduced by the iterative linear solver is not sufficient to overcome the gains obtained by compressing the linear system. We illustrate this behavior in Table 4.3 where we compare the performance of full factorization (0% compression rate) with that of preconditioner CP for problem 20TERM. We clearly see that the total solution times (denoted as θ_{tot}) obtained with full factorization are significantly shorter than those obtained with CP. Most notably, this trend holds for problems with up to 600,000 variables and the times scale linearly with the number of scenarios. These results illustrate that sparse direct factorization codes such as MA57 can efficiently handle certain large-scale problems. As we show in Section 4.4.2, this efficiency enables us to overcome scalability bottlenecks of Schur complement decomposition. Full factorization, however, will eventually become expensive as we increase the problem size and, at this point, the use of CP becomes beneficial. We illustrate this in Table 4.6 where we compare the performance of CP with that of full factorization for two large instances. Instance QSC2015 has 63,717 variables, while instance AUG3DC has 131,682 variables. We use θ_{fact} to denote the time spent in the factorization of the compressed matrix and of the block matrices, θ_{clus} to denote the time spent performing clustering operations, and θ_{gmres} to denote the time spent in GMRES iterations (without considering factorization operations in the preconditioner). As can be seen, the solution times of full factorization are dramatically reduced by using CP.

From Table 4.3 we can also see that the performance of CP deteriorates as we increase the compression rate. This is because the distortion metric increases as we increase the compression rate and thus the quality of the preconditioner deteriorates, as suggested by Theorems 4.3.1 and 4.3.2. We recall, however, that the bounds provided in these theorems depend on constants that change with the clustering parameters. Consequently, it is not obvious that reducing the distortion metric will improve the quality of the preconditioner. We designed a numerical experiment to gain more insight into this issue. In the experiment we compute the constants and metrics of Theorems 4.3.1 and 4.3.2 for two additional instances (GBD and LANDS) and for two different compression rates (50% and 75%). We only report the results at a single iteration because we observed similar behavior at other iterations. The results are summarized in Tables 4.4 and 4.5. As can be seen in Table 4.4, the constants c_K and $\sigma_{min}(\bar{Z})$ of Theorem 4.3.1 are insensitive to the compression rate. The distortion metric $\mathcal{D}_C$, on the other hand, changes by an order of magnitude. We also report the Schur complement error $\|E_Z\| = \|\bar{Z} - Z\|$ and we see that this error

Table 4.3: Effect of compression rates on 20TERM problem.

S	n	Compress.	IPit	LAit/IPit	θ_{tot}
100	76,463	0%	54	-	16
		50%	57	10	54
		75%	57	19	79
		87%	57	17	69
200	152,863	0%	69	-	44
		50%	72	9	137
		75%	72	14	166
		87%	72	18	185
400	305,663	0%	87	-	108
		50%	92	20	578
		75%	92	21	555
		87%	92	23	570
800	611,263	0%	88	-	232
		50%	97	25	1440
		75%	97	25	1427
		87%	97	27	1417

Table 4.4: Performance of different clustering strategies for benchmark problems (Theorem 1).

Problem	Compress.	c_K	$\sigma_{min}(\bar{Z})$	$\|Z-\bar{Z}\|$	$\mathcal{D}_C$
LANDS	50%	$6.8\times 10^{+2}$	3.7×10^{-3}	3.9×10^{-2}	6.5×10^{-2}
	75%	$6.8\times 10^{+2}$	3.8×10^{-3}	1.8×10^{-1}	5.8×10^{-1}
GBD	50%	$4.5\times 10^{+12}$	6.8×10^{-2}	6.6×10^{-3}	5.0×10^{-4}
	75%	$4.5\times 10^{+12}$	6.7×10^{-2}	6.1×10^{-2}	1.6×10^{-3}

Table 4.5: Performance of different clustering strategies for benchmark problems (Theorem 2).

Problem	Compress.	c_Z	$\mathcal{D}_C$	$\|(\bar{Z}-Z)Z^{-1}t_Z\|$	$\|(\bar{Z}-Z)t_Z\|$
LANDS	50%	$5.7\times 10^{+2}$	$1.0\times 10^{+1}$	$2.0\times 10^{+0}$	$6.1\times 10^{+2}$
	75%	$5.8\times 10^{+2}$	$1.0\times 10^{+2}$	$2.9\times 10^{+1}$	$8.5\times 10^{+3}$
GBD	50%	$9.2\times 10^{+0}$	1.0×10^{-1}	5.6×10^{-3}	9.2×10^{-2}
	75%	$9.2\times 10^{+0}$	3.5×10^{-1}	2.0×10^{-2}	7.1×10^{-1}

changes by an order of magnitude as well. In Table 4.5 we can see that the constant c_Z of Theorem 4.3.2 is insensitive to the compression rate but the distortion is rather sensitive as well. Moreover, we can see that metrics $\|(\bar{Z}-Z)Z^{-1}t_Z\|$, $\|(\bar{Z}-Z)t_Z\|$, and $\mathcal{D}_C$ change significantly with the compression rate. We highlight that the distortion metric of Theorem 1 is defined using the features (4.33) while the distortion metric of Theorem 2 is defined using the features (4.35). *From these results we can conclude that the distortion metrics proposed are indeed appropriate indicators of preconditioning quality and can thus be used to guide the construction of the preconditioners.*

From Table 4.3 we can also see that the deterioration of performance due to increasing compression rates becomes less pronounced as we increase the number of scenarios. The reason is that more redundancy is observed as we increase the number of scenarios and, consequently, compression potential increases. This behavior has been found in several instances and indicates that it is possible to deal with problems with a large number of scenarios.

In Table 4.6 we compare the performance of different clustering strategies. To this end, we perform clustering using features (4.33) (we label this as $X^{-1}V$) and using (4.37) (we label this as r_Z). As can be seen, the performance of both clustering strategies is very similar. This demonstrates that the design of the features (4.33) and (4.35) is consistent.

Table 4.6: Performance of different clustering strategies for benchmark problems.

Problem	Compress.	Clustering	IPit	θ_{tot}	θ_{fact}	θ_{clus}	θ_{gmres}	LAit	LAit/IPit
QSC205	0%		110	1331	1321				
	50%	$X^{-1}V$	110	220	157	5	42	747	6
	75%	$X^{-1}V$	110	91	25	6	45	933	8
	50%	r_Z	110	229	161	5	43	747	6
	75%	r_Z	110	89	24	5	43	924	8
AUG3DC	0%		11	1427	1423				
	50%	$X^{-1}V$	11	96	84	0.3	6	26	2
	75%	$X^{-1}V$	11	24	13	0.3	6	27	2
	50%	r_Z	11	93	80	0.3	6	26	2
	75%	r_Z	11	25	13	0.3	6	27	2

4.4.2 Stochastic Market Clearing Problem

We demonstrate the computational efficiency of the preconditioner by solving a stochastic market-clearing model for the entire Illinois power grid system (Pritchard et al., 2010; Zavala et al., 2010). The system is illustrated in Figure 4.1. The stochastic programming formulation is given by

$$\min_{x_i, X_i(\omega)} \sum_{i \in \mathcal{G}} \left(\beta_i x_i + \mathbb{E}_\omega \left[\beta_i^+ (X_i(\omega) - x_i)_+ - \beta_i^- (X_i(\omega) - x_i)_- \right] \right)$$

$$\text{s.t.} \quad \tau_n(f) + \sum_{i \in \mathcal{G}(n)} x_i = d_n, \;\; n \in \mathcal{N} \tag{4.46a}$$

$$\tau_n(F(\omega)) - \tau_n(f) + \sum_{i \in \mathcal{G}(n)} (X_i(\omega) - x_i) = 0, \quad n \in \mathcal{N}, \omega \in \Omega \tag{4.46b}$$

$$f, F(\omega) \in \mathcal{F}, \quad \omega \in \Omega \tag{4.46c}$$

$$(x_i, X_i(\omega)) \in \mathcal{X}_i(\omega), \quad i \in \mathcal{G}, \omega \in \Omega. \tag{4.46d}$$

Here, $\mathcal{N}$ denotes the set of network nodes and $\mathcal{L}$ the set of transmission lines. The set of all suppliers is denoted by $\mathcal{G}$. Subsets $\mathcal{G}(n)$ denote the set of players connected to node $n \in \mathcal{N}$. The forward (first-stage) dispatched quantities for players are x_i, and the spot (second-stage) quantities under scenario ω are $X_i(\omega)$. Symbol f denotes the vector of all line flows and $\tau_n(\cdot)$ are flow injections into node $n \in \mathcal{N}$. Similarly, $F(\omega)$ denotes the vector of line flows for each scenario ω. The demand is assumed to be deterministic and inelastic and is represented by $d_n,\ n \in \mathcal{N}$. The sets $\mathcal{F}$ and $\mathcal{X}_i(\omega)$ are polyhedral and define lower and upper bounds for the flows and dispatch quantities. The objective of the market clearing problem is to minimize the forward dispatch cost plus the expected recourse dispatch cost. Here $[y]_+ = \max\{y, 0\}$ and $[y]_- = \max\{-y, 0\}$. The coefficients β_i denote the supply price bids, and β_i^+ and β_i^- are price bids for corrections of the generators. A supplier i asks $\beta_i^+ > \beta_i$ to sell additional power or asks $\beta_i^- < \beta_i$ to buy power from the system (e.g., reduces output). The scenarios ω characterize the randomness in the model due to unpredictable supply capacities (in this case wind power). We use a sample-average approximation of the problem to obtain a deterministic equivalent.

The market clearing model has *large first-stage dimensionality.* The Schur complement has a dimension of 64,199 and has a large dense block. In Table 4.7 we

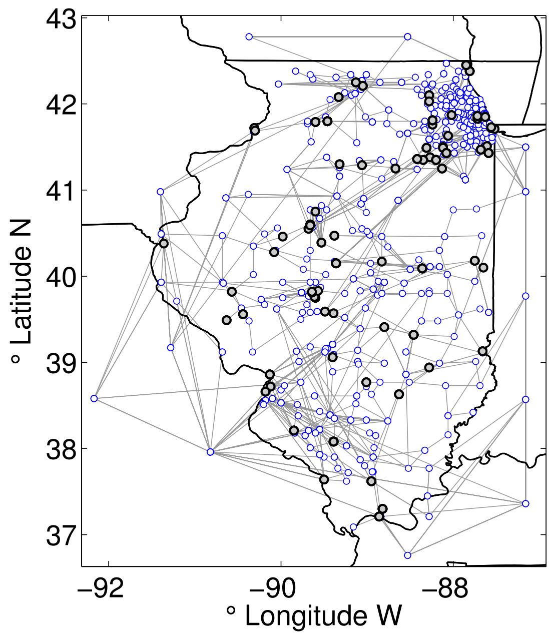


Figure 4.1.: Illinois transmission system. Dark dots are generation nodes and blue dots are demand nodes.

present the solution times for this problem using a Schur complement decomposition strategy for *a single scenario*. Here, θ_{tot} is the total solution time, $\theta_{factschur}$ is the time spent factorizing the Schur complement, $\theta_{formschur}$ is the time spent forming the Schur complement, and $\theta_{factblock}$ is the time spent factorizing the scenario blocks (in this case just one block). All times are reported in seconds. The solution time for this problem is 8.7 hr, with 13% of the time spent forming the Schur complement and 87% spent factorizing the Schur complement. Note that if more scenarios are added, the time spent forming and factorizing the Schur complement will dominate (even if the scenarios can be parallelized). Iterative strategies applied to the Schur complement system can avoid the time spent forming the Schur complement but not the factorization time because a preconditioner with a large dense block still needs to be factored (Petra and Anitescu, 2012).

We now assess the serial performance of CP. By comparing Tables 4.7 and 4.8 we can see that the full factorization approach will be as efficient as Schur complement decomposition for problems with up to 64 scenarios. In other words, it would be faster to factorize the full sparse KKT system than forming and factorizing the large Schur complement. The fast growth in solution time of the full factorization approach is remarkable, however. We attribute this to the tight connectivity induced by the network constraints which introduce significant fill-in. CP reduces the solution times of full factorization by a factor of 2 for the problem with 32 scenarios and by a factor of 8 for the problem with 64 scenarios. We highlight that CP is highly effective, requiring on average 5 to 10 GMRES iterations per IP iteration for compression rates of 50% and 11 to 13 iterations for compression rates of 75%. We also observe that the performance of different clustering strategies is similar.

For the problem with 64 scenarios we can see that the solution time of CP increases as we increase the compression rate from 75% to 87% (even if the factorization time is dramatically reduced). This is because the time spent in GMRES to perform backsolves and matrix-vector operations dominates the factorization time. We mitigate this by using the parallel implementation of CP. The results are presented in Table 4.9. We can see that the solution time spent in GMRES to perform backsolves and matrix-vector operations is dramatically reduced by exploiting the block-bordered-diagonal structure of the KKT matrix. *This enables us to solve a market clearing problem with over 1.2 million variables in 10 minutes, as opposed to 8 hours using the full factorization approach. This represents a speed up factor of 42.* By comparing the parallel results with those of Table 4.7 we can also see that the Schur complement approach is not competitive because of the time needed to form and factorize the Schur complement (this holds even for a single scenario).

From Table 4.9 we can see that scalability slows down as we increase the number of processes. This is because the remaining serial components (beyond backsolves and matrix-vector operations) of CP start dominating. This overhead includes operations

Table 4.7: Performance of Schur complement decomposition approach.

S	n	IPit	θ_{tot}	$\theta_{factschur}$	$\theta_{formschur}$	$\theta_{factblock}$
1	30,472	55	31280	27236	4023	4

Table 4.8: Serial performance of preconditioner CP against full factorization for stochastic market clearing problem.

S	n	Compress.	Cluster.	IPit	θ_{tot}	θ_{fact}	θ_{clus}	θ_{gmres}	LAit	LAit/IPit
16	309,187	0%		57	473	452				
		50%	$X^{-1}V$	57	544	119	0.4	325	631	11
		50%	r_Z	57	508	117	0.15	296	519	9
32	606,483	0%		65	3480	3414				
		50%	$X^{-1}V$	65	1477	661	8	606	574	8
		75%	$X^{-1}V$	65	1479	145	8	1141	1194	18
		50%	r_Z	65	1347	672	3	459	398	6
		75%	r_Z	65	1131	150	3	769	804	12
64	1,201,075	0%		64	28022	27883				
		50%	$X^{-1}V$	64	5163	3513	29	1292	660	10
		75%	$X^{-1}V$	64	2878	656	29	1844	902	14
		87%	$X^{-1}V$	64	2499	135	29	1990	1040	16
		50%	r_Z	64	5238	3492	12	1349	666	10
		75%	r_Z	64	3003	659	12	1924	937	14
		87%	r_Z	64	2440	115	12	1944	1147	17

Table 4.9: Parallel performance of preconditioner CP against full factorization for stochastic market clearing problem.

S	n	MPI Proc.	Compress.	IPit	θ_{tot}	θ_{fact}	θ_{clus}	$\theta_{factblock}$	θ_{gmres}	LAit	LAit/IPit
64	1,201,075	1	0%	64	**28022**	27883					
		1	87%	64	2440	115	12	288	1944	1147	17
		2	87%	64	1211	116	12	147	892	1025	16
		4	87%	64	817	134	12	80	592	919	14
		8	87%	64	**658**	152	12	44	398	905	14
		1	94%	64	3223	49	12	327	2764	1489	23
		2	94%	64	1558	43	12	151	1306	1471	22
		4	94%	64	993	49	12	84	801	1420	22
		8	94%	64	**733**	54	12	46	570	1409	22

inside the GMRES algorithm itself. We are currently investigating ways to parallelize these operations.

In Table 4.9 we also present experiments using a compression rate of 94%. We performed these experiments to explore the performance limit of CP. We can see that the performance of CP deteriorates in terms of total solution time because the number of GMRES iterations (and thus time) increases. Consequently, it does not pay off to cluster the KKT system further. We highlight, however, that the deterioration of CP in terms of GMRES iterations is *graceful*. It is remarkable that, on average, the linear system can be solved in 22 GMRES iterations when only four scenarios are used in the compressed matrix. This behavior again indicates that the computation of the second-stage variables in (4.23b) plays a key role in the performance of CP.

We emphasize on the efficiency gains obtained from parallelization with respect to the computation of the second-stage steps (4.23b). This step requires a factorization of all the block matrices K_s prior to calling the iterative linear solver. When the factorizations of the blocks are performed serially, the total solution time grows linearly with the number of scenarios. This can be observed from the block factorization times (denoted as $\theta_{factblock}$) reported in Table 4.9. In particular, the time spent in the factorization of the block matrices in the serial implementation (one processor) is a significant component of the total time. This overhead is eliminated using the parallel implementation (with almost perfect scaling).

4.5 Concluding Remarks

We have presented a preconditioning strategy for stochastic programs using clustering techniques. This inside-the-solver clustering strategy can be used as an alternative to (or in combination with) outside-the-solver scenario aggregation and clustering strategies. Practical features of performing inside-the-solver clustering is that no information on probability distributions is required and the effect of the data on the problem at hand is better captured. We have demonstrated that the preconditioners can be implemented in sparse form and dramatically reduce computational time compared to full factorizations of the KKT system. We have also demonstrated that the sparse form enables the solution of problems with large first-stage dimensionality that cannot be addressed with Schur complement decomposition. Scenario compression rates of up to 94 % have been observed in large problem instances.

5. NONLINEAR MODEL PREDICTIVE CONTROL OF A BATCH CRYSTALLIZATION PROCESS[1]

This chapter presents nonlinear model predictive control (NMPC) and nonlinear moving horizon estimation (MHE) formulations for controlling the crystal size and shape distribution in a batch crystallization process. MHE is used to estimate unknown states and parameters prior to solving the NMPC problem. Combining these two formulations for a batch process, we obtain an expanding horizon estimation problem and a shrinking horizon model predictive control problem. The batch process has been modeled as a system of differential algebraic equations (DAEs) derived using the population balance model (PBM) and the method of moments. Therefore, the MHE and NMPC formulations lead to DAE-constrained optimization problems that are solved by discretizing the system using Radau collocation on finite elements and optimizing the resulting algebraic nonlinear problem using IPOPT. The performance of the NMPC-MHE approach is analyzed in terms of setpoint change, system noise, and model/plant mismatch, and it is shown to provide better setpoint tracking than an open-loop optimal control strategy. Furthermore, the combined solution time for the MHE and the NMPC formulations is well within the sampling interval, allowing for real world application of the control strategy.

5.1 Preliminaries

Batch crystallization is a crucial process in the pharmaceutical industry because more than 90% of the active pharmaceutical ingredients (API) are in the form of

[1]Part of this section is reprinted with permission from “Real-time Feasible Multi-objective Optimization Based Nonlinear Model Predictive Control of Particle Size and Shape in a Batch Crystallization Process” by Cao, Y., Acevedo, D., Nagy, Z., and Laird, C.D., 2015. Submitted to Journal of Process control.

crystals (Alvarez and Myerson, 2010). The crystal size and shape distribution is of great concern to both product quality and downstream processing such as filtration. Primarily because of the technology limitations to monitor the crystal shape (Nagy et al., 2013), early works in the crystallization research community focused on modeling and controlling the size distribution of crystals (Qamar et al., 2009; Mesbah et al., 2009). Focused Beam Reflectance Measurements (FBRM) is frequently used to monitor the size distribution online (Braatz, 2002; Fujiwara et al., 2005; Puel et al., 2003). The last decade has witnessed a significant progress in monitoring and modeling the shape distribution of crystals allowing the standard feedback control (Nagy and Braatz, 2003; Wang et al., 2007; Wan et al., 2009; Patience and Rawlings, 2001; Mesbah et al., 2011, 2012). Derived using the multidimensional population balance model (PBM) (Hulburt and Katz, 1964; Ramkrishna, 2000) and the method of moments, the dynamic evolution of the crystal size and shape distribution can be modeled as a system of differential algebraic equations. The size and shape distribution can be controlled by manipulating the cooling profile of the reactor, which directly affects the supersaturation.

To balance the trade-off between the size and shape distribution, Acevedo et al. (2015) proposes a multi-objective optimization approach to control both the size and shape distribution offline. However, in the presence of model/plant mismatch and system noise, the real plant trajectory can be quite different from the optimal trajectory obtained from the open-loop multi-objective optimization. Therefore, in this chapter, we developed a nonlinear model predictive control (NMPC) formulation that can be used to control the crystal size and shape distribution in real-time and in the presence of modeling and measurement noise.

Linear MPC has been a popular advanced control strategy in industry for many years (Qin and Badgwell, 2003). Because of the advances in both computational power and optimization algorithms, nonlinear model predictive control (NMPC) has become more computational feasible and is more appropriate for inherently nonlinear systems to achieve higher product quality and satisfy tighter regulations (Rawlings, 2000; Mayne et al., 2000). The basic idea of NMPC is to solve an optimal control problem at each sampling instance with the updated measured or estimated states. The control values for only the next sampling instance are implemented and the entire process is repeated in the next sampling cycle. For batch processes, since our real interest is in the product quality at the end of the batch, end-point based shrinking horizon NMPC formulation is frequently used.

Nevertheless, for many processes, it is not possible (or cost effective) to accurately measure all states online, and model parameters may change from batch to batch. This challenge drives the need for a state estimator to reconstruct unknown states and parameters. The Extended Kalman filter (EKF) is a popular state estimator for unconstrained systems (Prasad et al., 2002). However, this technique is not appropriate for the batch crystallization model because of the highly nonlinear dynamics and hard constraints such as nonnegative concentrations. In contrast, nonlinear moving horizon estimation (MHE) uses nonlinear constrained optimization to estimate unknown states and parameters and has proven its advantages over EKF in many applications (Haseltine and Rawlings, 2005; Rao et al., 2003; Rawlings and Bakshi, 2006). Therefore, in this chapter, we propose an MHE formulation that can be used to estimate the unmeasured states in our model prior to solving the NMPC problem for the batch crystallization process.

The computational burden of this approach is that at each sampling instance, an expanding horizon estimation problem and a shrinking horizon model predictive control problem need to be solved. Both problems are DAE-constrained optimization problems and there exist multiple solution approaches. “Optimize then discretize” or indirect approaches try to solve the first-order optimality conditions for the DAE-constrained problem. For problems without inequality constraints, the first-order optimality conditions can be formulated as boundary value DAE problems. However, for problems with active inequality constraints, determining the switch points of the inequality constraints can become very challenging and thus limits the application of these methods. On the contrary, “discretize then optimize” or direct approaches discretize the control variables and solve the resulting nonlinear programming (NLP) problems. Among "discretize then optimize" approaches, the sequential approach discretizes only the control variables and treats the DAE system as a black box. A DAE integrator is used to simulate the system at each iteration and calculate its sensitivity with regards to the discretized control variables. One drawback of this approach is that the solution time increases significantly when the controls are discretized finer. However, a finer discretization of the controls can often improve the performance of the NMPC. In contrast, the simultaneous approach (Biegler, 2007; Biegler et al., 2002) discritizes both control and state variables and optimizes the resulting algebraic nonlinear problem with an NLP solver. The performance of the simultaneous approach is less dependent on the number of discretized control variables. Another advantage of this approach is that state constraints can be formulated in a more straightforward way. Therefore, this chapter chooses the simultaneous approach to solve these DAE-constrained optimization problems arising from the NMPC-MHE formulations.

One challenge of using the simultaneous approach is that the burden of manually discretizing the DAE system before it is embedded into an optimization formulation often lies on the user. However, packages such as the Modelica-based JModelica.org platform (Åkesson et al., 2010) allow for straightforward declaration of differential equations and automatically perform this transcription process. Therefore, we implement these control formulations for batch crystallization within the Modelica libarary, which is already interfaced with solvers like IPOPT.

This chapter is organized as follows: a description of the unseeded batch crystallization model is presented in Section 5.2. Section 5.3 presents the NMPC-MHE approaches and efficient methods to solve the related optimization problems. Section 5.4 demonstrates the performance of the NMPC-MHE compared with the open-loop control in terms of setpoint change, system noise, and model/plant mismatch. Final conclusions are presented in Section 5.5.

5.2 Multidimensional Unseeded Batch Crystallization Model

This section provides a brief description of the multidimensional unseeded batch crystallization model. The details can be found in Acevedo and Nagy (2014). The population balance model (PBM) has been widely used to describe the crystallization process. Considering only the effect of growth and nucleation, the population balance equation for a well-mixed batch crystallization process can be expressed as

$$\frac{\partial}{\partial t}n(t,X) + \nabla_X[Gn(t,X)] = B\delta(X - X_0) \tag{5.1a}$$

$$I.C. : n(0,X) = n_0(X)\,, \tag{5.1b}$$

where $n(t,X)$ is the density distribution at time t, X is the vector of characteristic lengths, G is the vector of growth rates, B is the nucleation rate, X_0 is the size of the nuclei, δ is the Dirac delta function acting at $X = X_0$, and $n_0(X)$ is the initial seed distribution. The population balance model can be transformed into a set of ordinary differential equations (ODEs) using the method of moments (MOM). If we only consider two characteristic dimensions, the length L and the width W of crystals, the moments can be expressed by

$$\mu_{ij} = \int_0^\infty \int_0^\infty n(t,X)W^iL^j dW dL\,. \tag{5.2a}$$

The ODEs obtained from the MOM with the assumption that the nucleus size is negligible, are given by

$$\frac{d\mu_{00}}{dt} = B \tag{5.3a}$$

$$\frac{d\mu_{10}}{dt} = G_1\mu_{00} \tag{5.3b}$$

$$\frac{d\mu_{01}}{dt} = G_2\mu_{00} \tag{5.3c}$$

$$\frac{d\mu_{11}}{dt} = G_1\mu_{01} + G_2\mu_{10} \tag{5.3d}$$

$$\frac{d\mu_{20}}{dt} = 2G_1\mu_{10}, \tag{5.3e}$$

where G_1 and G_2 are the growth rates along the width and length of the crystals respectively, and B is the nucleation rate. In this chapter, size independent growth rates and primary nucleation rate are considered as follows:

$$G_1 = k_{g_1} S^{g_1} \tag{5.4a}$$

$$G_2 = k_{g_2} S^{g_2} \tag{5.4b}$$

$$B = k_b S^b \tag{5.4c}$$

$$S = \frac{C - C_s(T)}{C_s(T)}\,, \tag{5.4d}$$

where the kinetic parameters $k_{g_1}, k_{g_2}, g_1, g_2, k_b$, and b are usually sensitive to process conditions. S is the relative supersaturation, C is the solute concentration, and C_s is the equilibrium concentration at a given temperature, which can be expressed using a polynomial expression, given by

$$C_s(T) = cT^2 + dT + e\,. \tag{5.5a}$$

According to the mass balance equation, the evolution of the solute concentrate is given by

$$\frac{dC}{dt} = -2\rho_c k_v G_1(\mu_{11} - \mu_{20}) - \rho_c k_v G_2 \mu_{20}\,, \tag{5.6a}$$

where ρ_c is the density of the solution and k_v is a constant volumetric shape factor.

5.3 Computationally Efficient Online NMPC-MHE

At the end of the batch crystallization process, the product qualities are evaluated in terms of the size and shape distribution of crystals. Therefore, the mean length

(ML) and aspect ratio (AR) are used to evaluate the quality of crystals. ML can be calculated with the following equation

$$ML = \frac{\mu_{01}}{\mu_{00}}\,, \tag{5.7a}$$

and AR is determined by the following equation

$$AR = \frac{\mu_{01}}{\mu_{10}}\,. \tag{5.8a}$$

The product qualities are determined by the supersaturation trajectory, which is dependent on the temperature profile. Thus, the temperature profile can be used to control the crystal qualities to achieve the desired size and shape distribution.

5.3.1 Off-line Multi-objective Optimization

Before implementing the control strategy, we need to set an endpoint target. To find achievable end-point setpoints for the ideal case, we first solve the multi-objective optimization problems off-line. The problems follow the formulations in Acevedo et al. (2015) and Ma et al. (2002),

$$\min_{u(t)} \ \varphi(y(t_f)) \tag{5.9a}$$

$$\text{s.t.} \ \frac{dz(t)}{dt} = f(z(t), u(t)) \tag{5.9b}$$

$$y(t) = c(z(t), u(t)) \tag{5.9c}$$

$$z(t_0) = \tilde{z_0} \tag{5.9d}$$

$$g(z(t), u(t)) \leq 0, t \in [t_0, t_f]\,, \tag{5.9e}$$

where t is the time, t_0 and t_f are the start time and end time of the process, $z(t)$ is the vector of state variables including differential variables and algebraic variables, $y(t)$ is the vector of output variables AR, ML and C, and u represents the manipulated variable temperature. The initial state values $\tilde{z_0}$ of the process is known. Equation

(5.9e) is a vector of constraints on the inputs and state variables that can be further detailed using the following set of equations:

$$T_{min} <= T(t) <= T_{max} \tag{5.10a}$$

$$-R_{max} <= \frac{dT(t)}{dt} <= 0, \tag{5.10b}$$

$$C(t_f) - C_{max} <= 0\,. \tag{5.10c}$$

Equation (5.10a) and (5.10b) ensure that changes of temperature are within the operation range. Equation (5.10c) is the yield constraint.

For batch crystallization process, we want to avoid needle crystals. Therefore, we want AR to be small and ML to be large. The objective function is defined as $(1-w)AR - wML$. With a set of weight values $0 < w < 1$, we can calculate a set of non-dominated points. Without the existence of any model/measurement noise or model/plant mismatch, each pareto point is achievable using either the open-loop control or NMPC. Therefore, We should choose points on or above the pareto front as the setpoint.

5.3.2 Endpoint-based Shrinking Horizon NMPC Formulation

NMPC uses a nonlinear model to predict the dynamic behavior of the system and thus to determine the optimal input profile. This section describes the endpoint based shrinking horizon NMPC used in this chapter. The whole process interval [t_0,

t_f] can be discretized into N steps. At a sampling instance t_k, the following optimal control problem is solved online:

$$\min_{u(t)} \quad \|y(t_f) - y_{set}\|^2_{\Pi} \tag{5.11a}$$

$$\text{s.t.} \quad \frac{dz(t)}{dt} = f(z(t), u(t)) \tag{5.11b}$$

$$y(t) = c(z(t), u(t)) \tag{5.11c}$$

$$z(t_k) = \hat{z}(t_k) \tag{5.11d}$$

$$g(z(t), u(t)) \leq 0, t \in [t_k, t_f]\,, \tag{5.11e}$$

where $\hat{z}(t_k)$ is a vector of states at t_k estimated from measurements using MHE and y_{set} is the setpoint we want to achieve at the end of the batch based on the off-line multi-objective optimization. Although the whole input profile in the interval [t_k, t_f] is computed, only the control action in the interval [t_k, t_{k+1}) is implemented. At the next sampling instance t_{k+1}, the control horizon shrinks from [t_k, t_f] to [t_{k+1}, t_f], and the optimal control problem is re-evaluated with new measurements and updated state estimates.

For end-point based NMPC, the objective function only depends on $z(t_f)$. Here, we want to minimize the deviation of the product quality at the end of the batch from the desired product qualities. The deviation is weighted by a positive definite matrix O.

5.3.3 Nonlinear Expanding Horizon MHE formulation

Since the NMPC approach requires that the initial values of all state variables $\hat{z}(t_k)$ are known, and since not all states can be measured , we need to reconstruct the state information from a limited number of measurements. At each sampling instance t_k, a state estimation problem is solved online and the solution $\hat{z}(t_k)$ is provided to the NMPC formulation. The objective function of the optimization problem penalizes the model noise and the deviation of predicted output from the measurements and the estimated parameters from the reference value. The formulation of the estimation problem is as follows:

$$\min_{p,w(t)} \int_{t_0}^{t_k} \| w(t) \|_R^2 dt + \sum_{i=1}^{k} \| y(t_i) - y(t_i)^{meas} \|_W^2 + \|p - p^{ref}\|_Z^2 \tag{5.12a}$$

$$\text{s.t.} \quad \frac{dz(t)}{dt} = f(z(t), u(t), p) + w(t) \tag{5.12b}$$

$$y(t) = c(z(t)) \tag{5.12c}$$

$$z(t_0) = \tilde{z_0} \tag{5.12d}$$

$$z(t) \geq 0, t \in [t_0, t_k] , \tag{5.12e}$$

where $y(t_i)^{meas}$ is their corresponding set of actual measured values at sampling instance t_i, w is the vector of model noise, p is the vector of parameters selected for online adjustment, p^{ref} is the vector of reference value for p, and R, W and Z are weighting matrices. Generally, the objective function of MHE also includes a term about the initial state estimation. However, for this unseeded batch process, the initial state values are known. The time span of this problem is $[t_0,t_k]$, therefore the entire historical input profile is known. At the next sampling instance t_{k+1}, the time span of the problem expands from $[t_0,t_k]$ to $[t_0,t_{k+1}]$, and the estimation problem is re-evaluated with new measurements $y(t_{k+1})$.

5.3.4 Efficient Optimization via the Simultaneous Approach

The off-line multi-objective optimization problem, the NMPC problem and MHE problem are all DAE-constrained optimization problems. These problems can be solved using simultaneous method by discretizing the problem with collocation methods and optimizing the large NLP problem afterwards. The details of this method can be found in Biegler (2010). This section uses the NMPC problem as an example to give a brief description of this method. This approach partitions the time domain $[t_k,t_f]$ into n_e stages with length h_i, $i = 1, ..., n_e$, where $\sum_{i=1}^{ne} h_i = t_f - t_k$.

At each stage, we discretize using n_c collocation points. This section assumes Radau collocation is used. After discretization, the problem can be formulated as:

$$\min_{u_{i,j},z_{i,j},y_{i,j},\dot{z}_{i,j}} \quad \|z_{n_e,n_c} - z_{set}\|_{\Pi}^2 \tag{5.13a}$$

$$\text{s.t.} \quad z_{i,j} = z_i + h_i \sum_{k=1}^{n_c} w_{j,k}\dot{z}_{i,j} \tag{5.13b}$$

$$\dot{z}_{i,j} = f(z_{i,j}, u_{i,j}) \tag{5.13c}$$

$$y_{i,j} = c(z_{i,j}, u_{i,j}) \tag{5.13d}$$

$$z_1 = \hat{z}(t_k) \tag{5.13e}$$

$$z_{i+1} = z_{i,n_c} \tag{5.13f}$$

$$g(z_{i,j}, u_{i,j}) \leq 0 \tag{5.13g}$$

$$\forall i = 1, ..., n_e, j = 1, ...n_c\,, \tag{5.13h}$$

where w are the coefficients from the radau collocation method.

One challenge of using the simultaneous approach is manual discretization of the problem. However, the Modelica-based JModelica.org platform (Åkesson et al., 2010) allows for straightforward declaration of differential equations and automatically perform this transcription process using direct collocation methods. Therefore, we implement the control formulation for batch crystallization within the Modelica libarary, which is already interfaced with solvers like Ipopt.

5.4 Results and Discussion

In the results shown later, the potassium dihydrogen phosphate (KH_2PO_4, KDP) and water system is used as a case study. The equilibrium concentration C_s for KDP estimated by Togkalidou et al. (2000, 2001) is given by

$$C_s(T) = 9.3027 \times 10^{-5}T^2 - 9.7629 \times 10^{-5}T + 0.2087\,, \tag{5.14}$$

where the unit of T is °C and the unit of C_s is g/cm^3. The kinetic parameters of KDP Gunawan et al. (2002); Majumder and Nagy (2013); Acevedo and Nagy (2014) and process conditions are summarized in Table 5.1.

Table 5.1: Parameters used in the control of unseeded cooling batch crystallization systems

Parameters	Values	Units	Parameters	Values	Units
k_{g_1}	0.073	cm/min	T_{max}	45	°C
g_1	1.48	Dimensionless	T_{min}	5	°C
k_{g_2}	0.60	cm/min	R_{max}	-4	°C/min
g_2	1.74	Dimensionless	C_0	0.395	g/cm^3
k_b	$4.494 \cdot 10^6$	#/cm^3 min	C_{max}	0.2	g/cm^3
b	2.04	Dimensionless	t_f	90	min
k_v	0.67	Dimensionless	ρ_c	2.34	g/cm^3

Before implementing the control strategies, we first find achievable end-point setpoints by solving open-loop multi-objective optimization problems offline. Two cases are considered with the control variable T discretized with 6 control steps and 90 control steps. The temperature profile inside the same step is assumed to be linear. The state variables are discretized with 90 steps. For the larger case (using 90 control steps), the number of variables and constraints in each problem are 5320 and 5230 respectively. All the problems are initialized using the simulation data with linear temperature profile. The solution time for Ipopt on an individual pareto point is approximately 2 seconds. By solving the problem repeatedly with different weights we obtain the pareto front. Figure 5.1 clearly demonstrate the trade-off between the two objectives. The front obtained using 6 control steps is worse than that using 90 control steps since it has fewer degrees of freedom.

All the points below the pareto front are not achievable even in the ideal circumstance with no system noise or model/plant mismatch. Therefore we should choose a point above the pareto front as the setpoint, which is then used to construct the

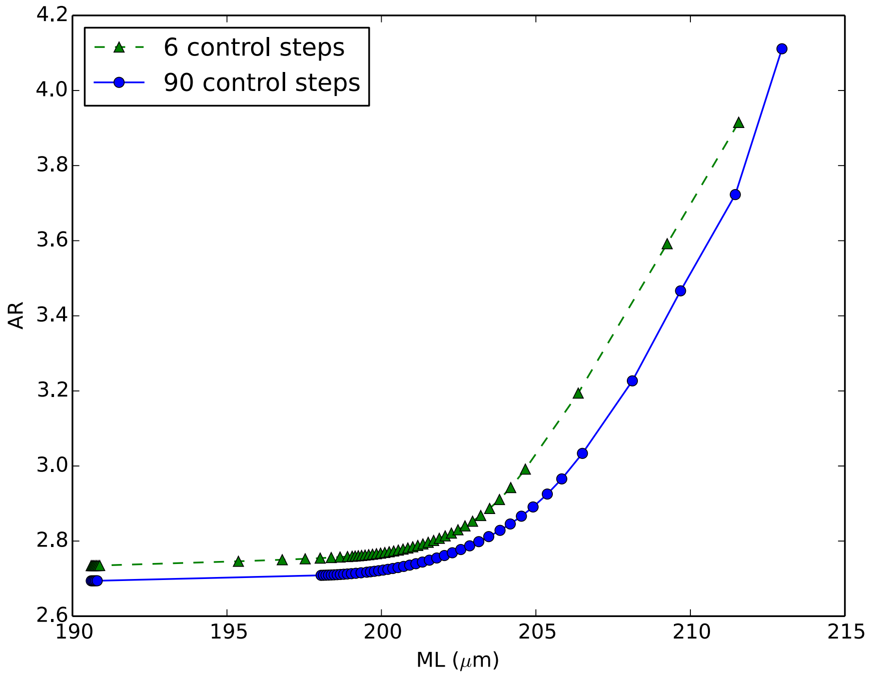


Figure 5.1.: Pareto fronts between AR and ML using 6 and 90 control steps

objective function in the NMPC. The objective function used in the NMPC for this application is

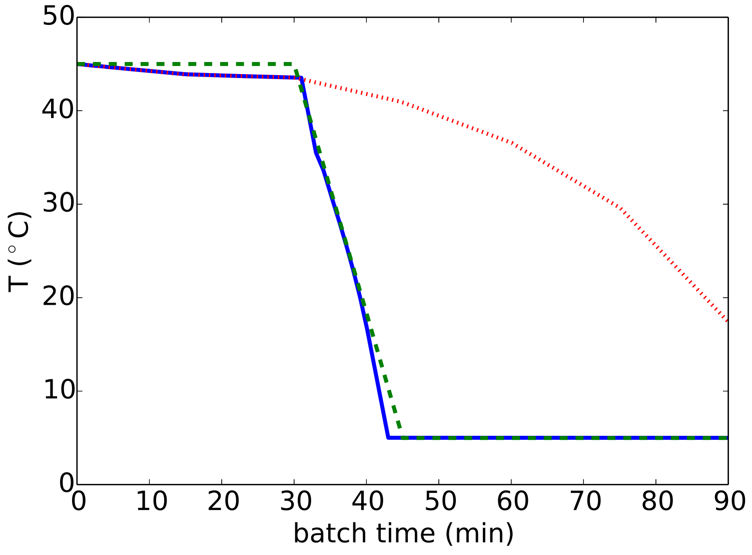
$$cost = 100(AR(t_f) - AR_{set})^2 + (ML(t_f) - ML_{set})^2, \tag{5.15a}$$

where AR_{set_1} and ML_{set_2} are the end-point setpoints. This *cost* function is also used to judge the performance of different control approaches. We analyze the performance of the NMPC-MHE during setpoint change, system noise and model/plant mismatch.

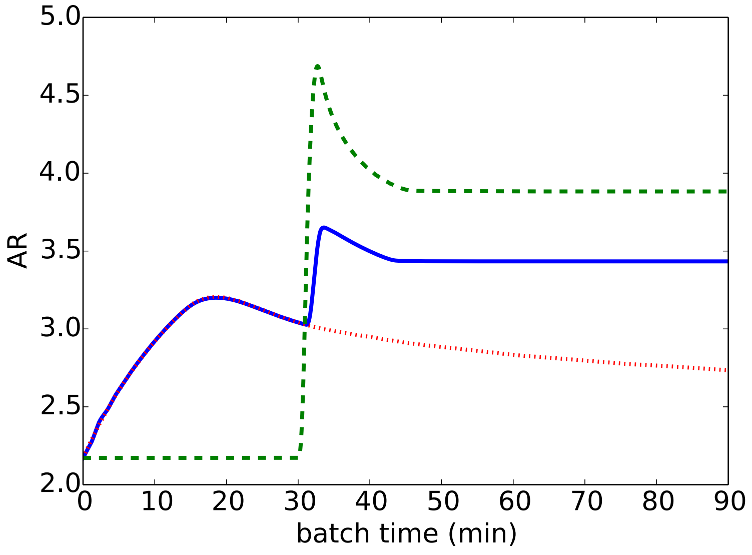
5.4.1 Setpoint Change

Although setpoint change during the batch process is uncommon, we use this study to examine the performance of our closed-loop NMPC-MHE. We first select two points on the pareto front with 6 control steps as setpoints so that it is more fair to compare the performance with different control steps. Here we choose $AR_{set_1} = 2.735, ML_{set_1} = 190.53\mu$m for setpoint s_1 and $AR_{set_2} = 3.883, ML_{set_2} = 210.68\mu$m for setpoint s_2. At a certain time during the process, the setpoint is changed from s_1 to s_2. For the results discussed in this subsection, we assume all states are exactly measured and there is no system noise and model/plant mismatch.

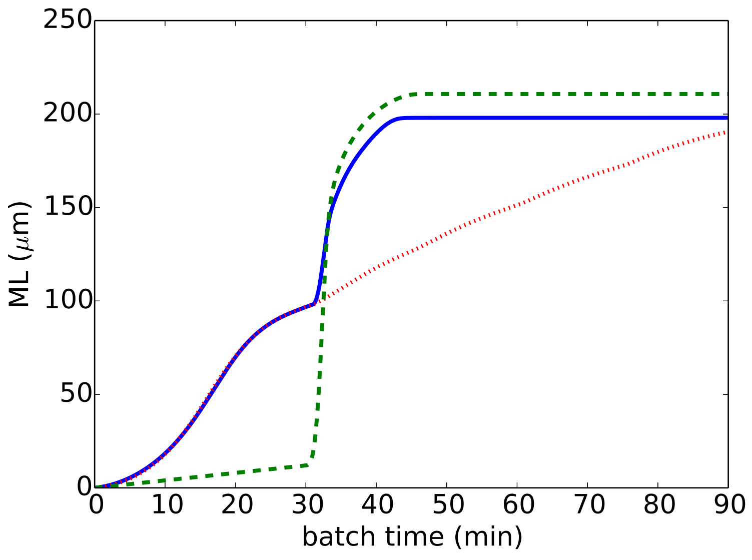
Figure 5.2 shows the input and measurement profiles when the setpoint is changed at t = 30 min. The number of control and sampling steps are all 90. Before the setpoint change, the NMPC profile follows the open-loop trajectory for achieving s_1. However, after the setpoint change, NMPC profile moves closer to the open-loop trajectory for achieving s_2.



T (°C)
batch time (min)



AR
batch time (min)



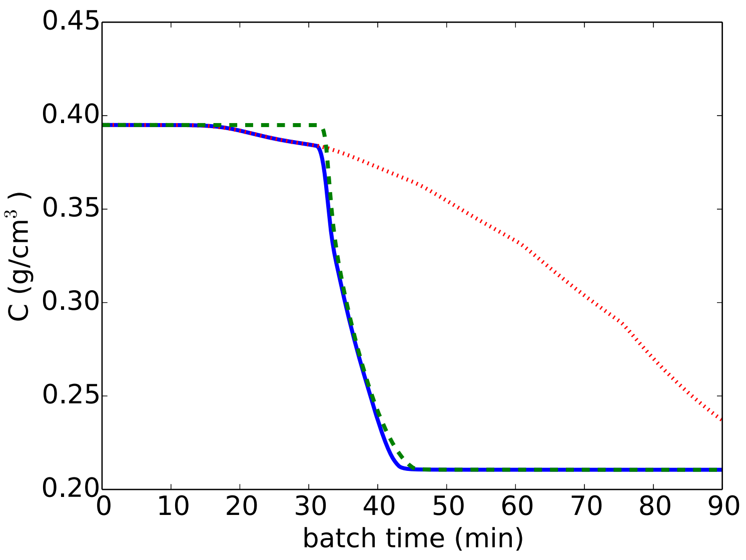


Figure 5.2.: Input and measurement profiles when the setpoint is changed at t=30 min. The solid line denotes the NMPC profile, the dotted line denotes the open-loop trajectory achieving endpoint setpoint s_1, and the dashed line denotes the open-loop trajectory achieving endpoint setpoint s_2. Before t=30 min, the NMPC profile follows the dotted line, while after setpoint change, the NMPC profile moves closer to the dashed line.

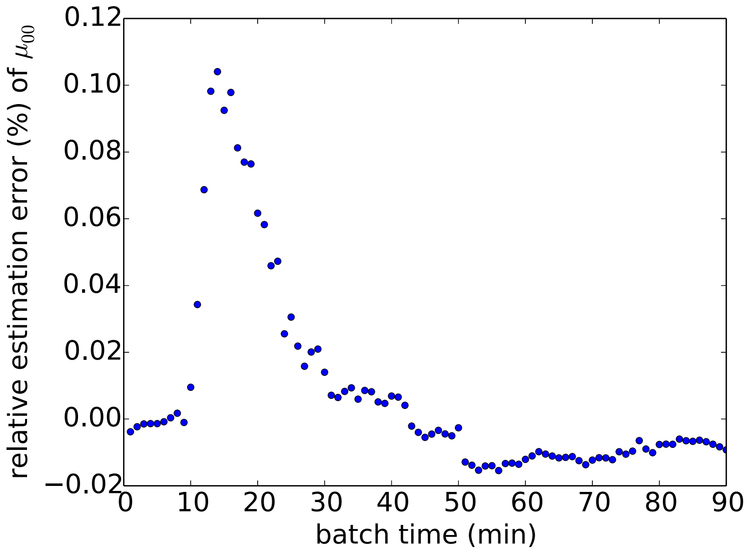
Table 5.2: Effect of t_{change} and sampling/control steps on end-point performance.

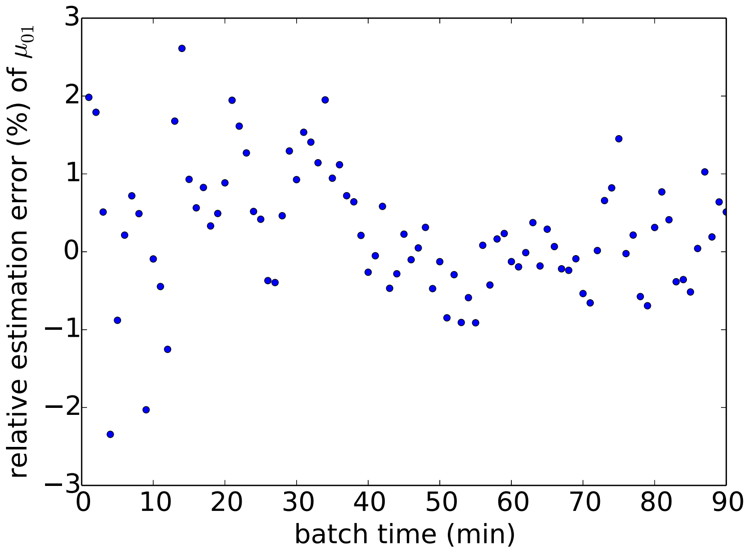
Sampling/Control Steps	t_{change} (min)	AR	ML(μm)	$cost$
6	0	3.880	210.65	0.00
	30	3.358	198.10	185.8
	60	2.884	195.32	335.7
	90	2.735	190.53	537.8
18	0	3.883	210.66	0.0
	30	3.533	197.67	181.5
	60	2.784	197.26	300.8
	90	2.735	190.53	537.8
90	0	3.882	210.65	0.0
	30	3.439	197.97	181.2
	60	2.769	197.47	298.6
	90	2.735	190.53	537.8

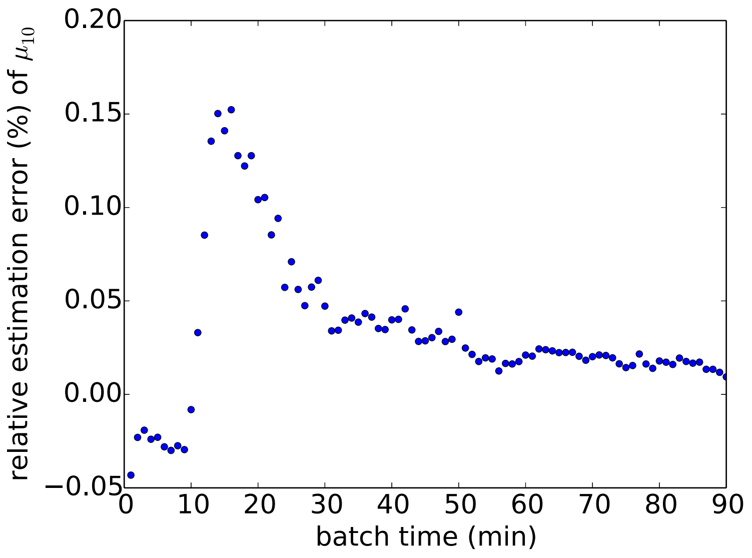
Table 5.2 shows the effect of t_{change} and sampling/control steps, where t_{change} is the time setpoint changed. The endpoint product quality of NMPC is closer to setpoint s_2 when the setpoint change is performed earlier in the process. It also indicates that increasing the number of sampling/control intervals can improve the performance of NMPC slightly in this case.

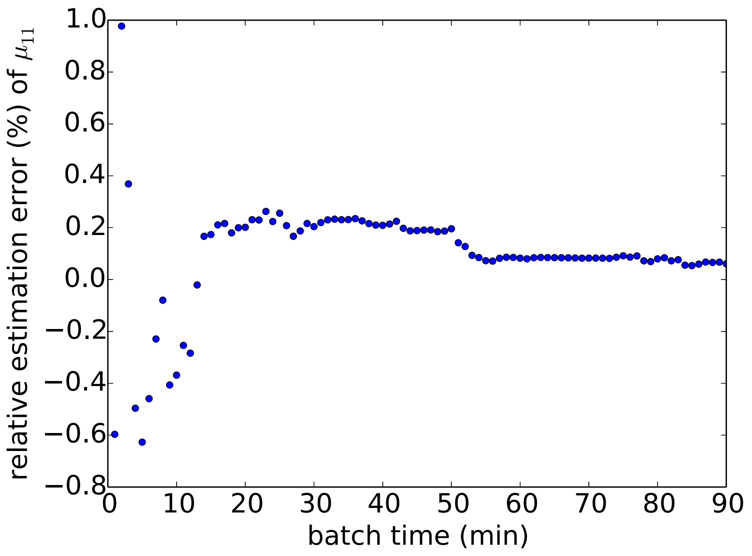
5.4.2 System Noise

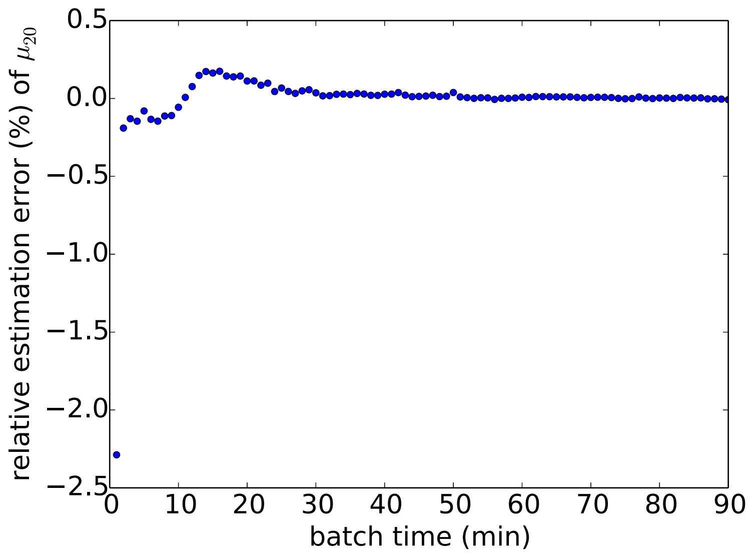
This subsection demonstrates the effectiveness of the closed-loop MHE-MPC for the batch process with both model and measurement noise. We assume that there is one model noise term w added to $\frac{d\mu_{01}(t)}{dt}$ and the noise follows truncated normal distribution on the interval $[-20\ \ 20]$cm/cm^{3}min. The mean and standard deviation of the original normal distribution is 0 and 10 cm/cm^{3}min respectively. We also assume that the measurement noise corresponding to three measurements ML, AR and C follow a truncated normal distribution on the interval $[-6\ \ 6]\mu$m, $[-0.1\ \ 0.1]$, and $[-0.004\ \ 0.004]$g/cm^3. The mean values of the original normal distribution are all zero, and the standard deviations are 3 μm, 0.05, and 0.002 g/cm^3, respectively. Because of the noise, points on the pareto front can no longer be achieved. Therefore, we consider a more conservative setpoint s_3, where AR_{set_3} is 2.9 and ML_{set_3} is 195 μm.






relative estimation error (%) of μ_{10}
batch time (min)
0.20
0.15
0.10
0.05
0.00
−0.05
0
10
20
30
40
50
60
70
80
90


relative estimation error (%) of μ_{11}
batch time (min)
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
10
20
30
40
50
60
70
80
90



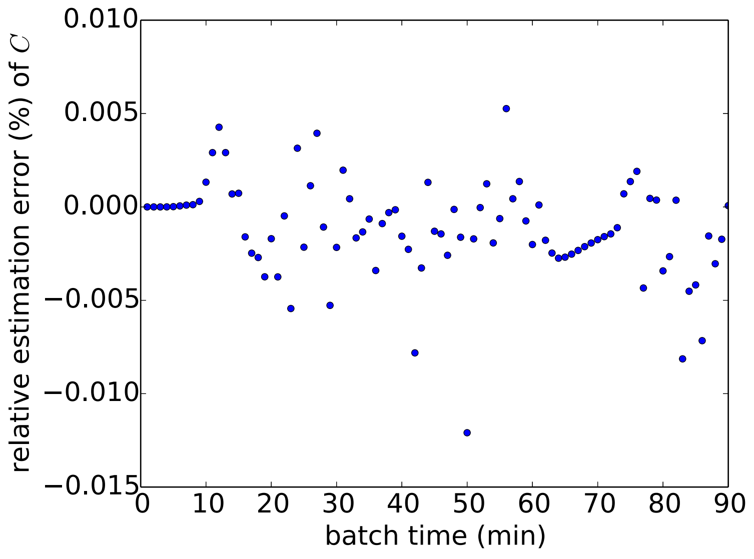


Figure 5.3.: Evolution of the relative estimation error of states using MHE with 90 control and sampling steps.

Figure 5.3 shows the relative estimation error of states using MHE with 90 control and sampling steps for one noise scenario. This figure shows that MHE can reconstruct the evolution of the state for this batch process fairly accurately.

Table 5.3: Performance of NMPC-MHE (value of *cost*) on 10 cases with model and measurement noise. Closed loop with true states is the performance of NMPC with 90 control and sampling steps and all states exactly measured.

Scenario No.	Open Loop (*cost*)	Closed Loop (*cost*)			
	90steps	6 steps	18 steps	90 steps	True States
1	159	31.4	24.8	18.2	17.7
2	14.5	1.0	0.9	1.0	1.1
3	105.5	33.1	40.7	40.6	42.9
4	52.4	27.7	17.0	12.0	12.7
5	297.2	200.1	154.4	140.4	116.9
6	53.0	22.5	17.8	9.7	10.5
7	28.4	1.4	3.0	3.8	2.7
8	44.5	16.4	8.5	3.9	2.7
9	81.4	12.6	10.7	9.3	9.4
10	13.7	24.8	8.9	1.1	3.2
Average	85.0	37.1	28.7	23.9	22.0

Table 5.3 highlights the performance improvement that can be achieved from the NMPC-MHE approach. It shows that the performance of ideal NMPC with all states accurately measured is much better than that of the open-loop control. The performance of NMPC-MHE is very close to that of the NMPC with true states and is also much better than that of the open-loop control. This observation is only possible with the excellent performance of MHE to reconstruct unmeasured states and thus give accurate feedback. This table also indicates that increasing the number of control and sampling steps can greatly improve the performance of NMPC-MHE.

Figure 5.4 shows that the CPU computational time of the expanding horizon estimation problems increases along the batch process, and that of the shrinking horizon model predictive control problems decreases. Because of the efficient computational framework used, the maximum total computation time (approximately 7 seconds) is far below the sampling interval of 60 seconds, allowing for real world application of the proposed control strategy.

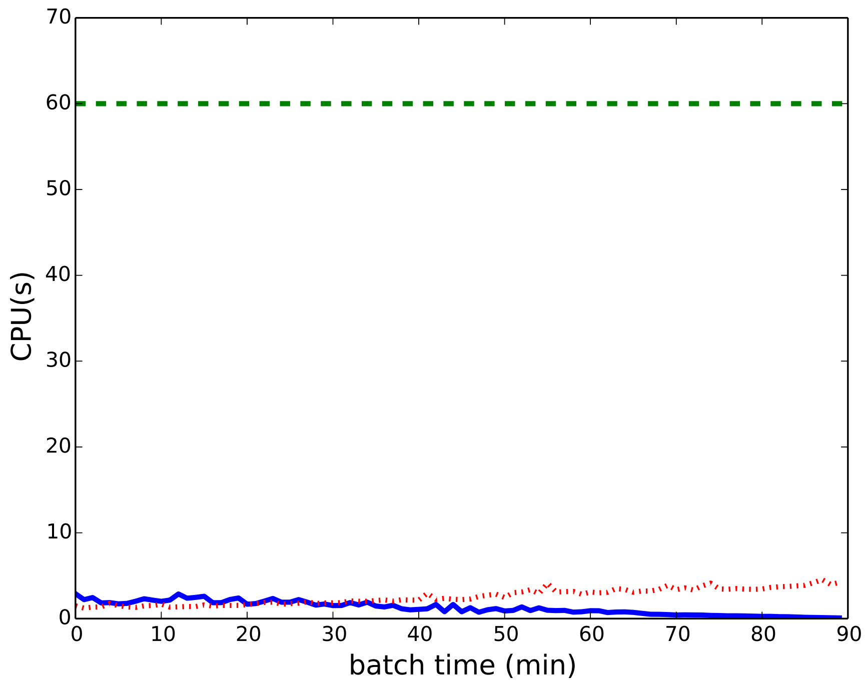


Figure 5.4.: Computational time of NMPC (solid line), computational time of MHE (dotted line), and sampling interval (dash line) along the batch with 90 control and sampling steps.

5.4.3 Model/Plant Mismatch

This section considers the case with both model/plant mismatch and measurement noise. It is assumed that the actual value in the plant for the parameter kb is $5.494{\cdot}10^6$, while the initial guess used in the first MPC instance is $4.494 \cdot 10^6$. By using the MHE, we not only reconstruct the state profiles from measurements, but also estimate the unknown parameter. The unknown parameter and measurement noise become degrees of freedom in the MHE. Another term $\| kb - 4.494 \cdot 10^6 \|^2$ is added to the objective function for regularization inside the MHE. The measurement noise and setpoint are the same as that in the Section 5.4.2.

Table 5.4: Performance of NMPC-MHE (value of *cost*) on 10 cases with model/plant mismatch and measurement noise. Closed loop with 6 steps, 18 steps, and 90 steps are the performance of NMPC with state estimation and parameter updates from MHE. Closed loop with true states is the performance of NMPC with 90 control and sampling steps and all states exactly measured. However, the parameter kb is fixed to be $4.494 \cdot 10^6$, which is not accurate.

Scenario No.	Open Loop (*cost*)	Closed Loop (*cost*)			
	90steps	6 steps	18 steps	90 steps	True States
1	73.5	7.7	7.6	3.4	6.8
2	73.5	5.3	7.5	4.5	6.8
3	73.5	7.4	7.5	2.9	6.8
4	73.5	5.2	4.0	4.0	6.8
5	73.5	8.6	9.7	3.5	6.8
6	73.5	8.6	5.0	3.7	6.8
7	73.5	7.3	5.6	3.3	6.8
8	73.5	6.4	7.6	3.8	6.8
9	73.5	7.7	7.7	4.9	6.8
10	73.5	7.3	7.6	5.9	6.8
Average	73.5	7.2	7.0	4.0	6.8

The estimated parameter and state profiles from MHE are used in the NMPC at the same sampling instance. Therefore, the accuracy of the state profile and parameter estimation is very important to the performance of NMPC-MHE. The state profile estimation results are still as accurate as the the results shown in Section 5.4.2. However, the parameter estimation, as shown in Figure 5.5, is not very accurate over the first 15 minutes but gradually converges to the true value as more measurement data becomes available.

Table 5.4 shows the overall performance of NMPC-MHE in dealing with model/plant mismatch. Again, the performance of NMPC-MHE is much better than that of open-

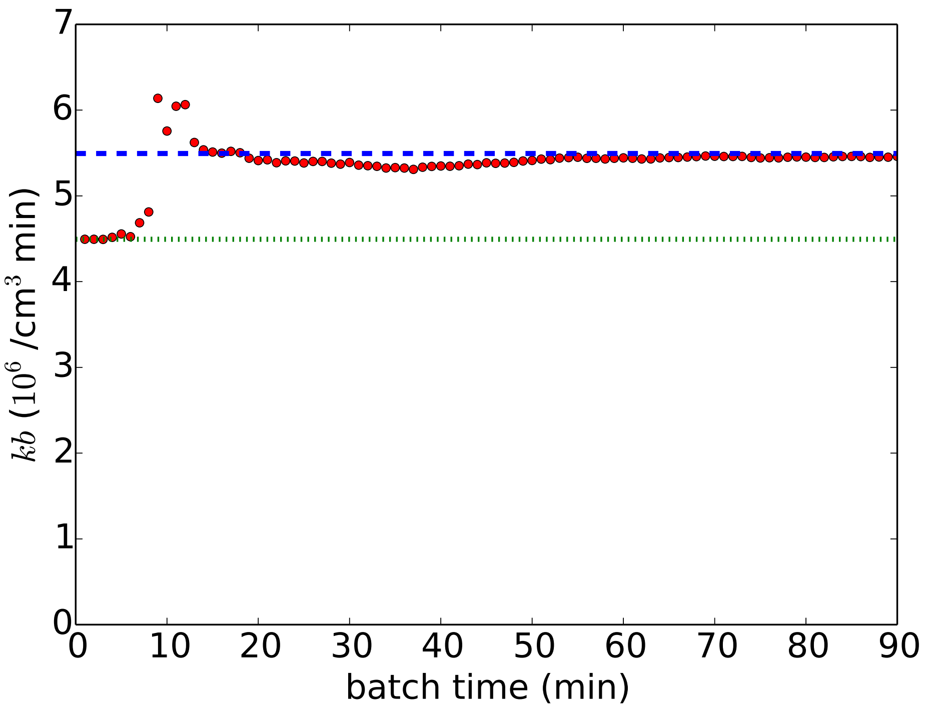


Figure 5.5.: Actual value (dash line), initial guess(dotted line), and MHE estimation (dots) of parameter kb along the batch process with 90 control and sampling steps.

loop control. Increasing the number of control and sampling steps can improve the performance of NMPC-MHE in the presence of model/plant mismatch. This table also considers the performance of NMPC with all states accurately measured and a fixed initial guess (inaccurate) of the parameter kb. Comparing with NMPC-MHE, this NMPC has exact state measurement but has no parameter estimation updates. The performance of NMPC-MHE is slightly better than that of the NMPC with true states, indicating the importance of parameter updates.

5.5 Concluding Remarks

In summary, we have developed a computationally effective NMPC-MHE formulations for batch crystallization processes to control the crystal size and shape distribution. At each sampling instance, we need to solve an expanding horizon estimation problem and a shrinking horizon model predictive control problem. Based on a nonlinear DAE model, the estimation problem estimates unknown states and parameters and the control problem determines the optimal input profiles. Both DAE-constrained optimization problems are solved by discretizing the system using Radau collocation and optimizing the resulting algebraic nonlinear problem. We build these formulations in the Modelica modeling language to support solution through the JModelica modeling and optimization framework. This framework performs automatic transcription, and it is already interfaced with IPOPT.

The performance of this control strategy was tested using a case study of a 90-minute batch crystallization process with 90 control steps and sampling steps. It was analyzed in terms of setpoint change, system noise, and model/plant mismatch. For all cases, the performance of the NMPC-MHE is shown to provide better setpoint tracking than the open-loop optimal control strategy. The combined solution time for the MHE and the NMPC formulations is well within the sampling interval, allowing for real world application of the control strategy.

6. ROBUST NONLINEAR MODEL PREDICTIVE CONTROL OF A BATCH CRYSTALLIZATION PROCESS

The quality of the NMPC approach described in the previous chapter depends on the accuracy of the underlying model. Despite the high fidelity of using the nonlinear models based on the first principles, there are still uncertainties associated with external and internal disturbances. Although the robust Input-to-State Stability (ISS) of NMPC can be proven for ideal NMPC (Jiang and Wang, 2001; Magni and Scattolini, 2007) under several assumptions, it is of limited use in analyzing the robust performance of batch processes.

Several approaches have been proposed to take those uncertainties into consideration in the design of NMPC. The most widely-studied approach is to solve a min-max optimization to minimize the worst case at each sampling instance (Scokaert and Mayne, 1998). One concern about this approach is that the nominal performance is sacrificed as the min-max optimization chooses a very conservative control strategy. Some authors proposed to minimize the expected value of the performance index based on multiple uncertainty scenarios (Huang et al., 2009). However, this approach does not consider the variance of the performance index. Nagy and Braatz (2003) proposes one formulation to minimize a weighted sum of expected value and variance of the performance index. While all of these approaches can be implemented within a feedback framework, this feedback is not considered in the NMPC optimization formulation itself. By contrast, Magni et al. (2003) optimizes the control laws instead of the control steps at each sampling step. However, if the form of the control law is overly complex, this approach may not be computationally feasible.

In this chapter we will use the min-max robust NMPC to deal with uncertainties arising from the batch crystallization process.

6.1 Robust NMPC formulation

For a batch process controlled by the min-max robust NMPC, at each sampling instance t_k, instead of solving the problem 5.11, the following min-max optimal control problem is solved online.

$$\min_{u(t)} \quad worst \tag{6.1a}$$

$$\text{s.t.} \quad worst \geq \|y_s(t_f) - y_{set}\|_{\Pi}^2 \tag{6.1b}$$

$$\frac{dz_s(t)}{dt} = h(z_s(t), u(t), p_s) \tag{6.1c}$$

$$y_s(t) = c(z_s(t), u(t)) \tag{6.1d}$$

$$z_s(t_k) = \hat{z}(t_k) \tag{6.1e}$$

$$g(z_s(t), u(t)) \leq 0, \tag{6.1f}$$

$$t \in [t_k, t_f], \forall s \in \mathcal{S}, \tag{6.1g}$$

where z_s is a vector of states if the real parameter turn out to be $p = p_s$. The control profile u needs to be determined before the realization of p. Hence, we can view u and $worst$ as the first stage variables and z_s and y_s as the second stage variables.

This DAE constrained problem can be discretized using collocation methods. This approach partitions the time domain $[t_k,t_f]$ into n_e stages with length h_i, $i = 1, ..., n_e$, where $\sum_{i=1}^{ne} h_i = t_f - t_k$. At each stage, we discretize using n_c collocation points. This section assumes that Radau collocation is used. After discretization, the problem can be formulated as:

$$\min_{u^{i,j}, z^{i,j}, y^{i,j}, \dot{z}^{i,j}} \quad worst \tag{6.2a}$$

$$\text{s.t. } worst \geq \|y_s^{n_e,n_c} - y_{set}\|_\Pi^2 \tag{6.2b}$$

$$z_s^{i,j} = z_s^i + h_i \sum_{k=1}^{n_c} w_{j,k} \dot{z}_s^{i,j} \tag{6.2c}$$

$$\dot{z}_s^{i,j} = h(z_s^{i,j}, u^{i,j}) \tag{6.2d}$$

$$y_s^{i,j} = c(z_s^{i,j}, u^{i,j}) \tag{6.2e}$$

$$z_s^1 := \hat{z}(t_k) \tag{6.2f}$$

$$z_s^{i+1} := z_s^{i,n_c} \tag{6.2g}$$

$$g(z_s^{i,j}, u^{i,j}) \leq 0 \tag{6.2h}$$

$$\forall i = 1, ..., n_e, j = 1, ...n_c, s \in \mathcal{S} \tag{6.2i}$$

where w are the coefficients from the radau collocation method.

6.2 Efficient Parallel Algorithm via the Explicit Schur Complement Decomposition

If we view $worst$ and $u^{i,j}$ as first stage variables, and $z_s^{i,j}, y_s^{i,j}$, and $\dot{z}_s^{i,j}$ as second stage variables, the above problem fits problem formulation (6.3) of the two stage stochastic programs:

$$\min f_0(x_0) + \sum_{s \in \mathcal{S}} f_s(x_s, x_0) \tag{6.3a}$$

$$\text{s.t. } c_0(x_0) = 0 \qquad (\lambda_0) \tag{6.3b}$$

$$c_s(x_0, x_s) = 0 \qquad (\lambda_s), \quad s \in \mathcal{S} \tag{6.3c}$$

$$x_0 \geq 0 \qquad (\nu_0) \tag{6.3d}$$

$$x_s \geq 0 \qquad (\nu_s), \quad s \in \mathcal{S}. \tag{6.3e}$$

Here, x_s is the second stage variable for scenario s, $\lambda_0 \in \Re^{m_0}$ and $\nu_0 \in \Re^{n_0}$ are the dual variables for the first stage equality constraints and the bounds, and $\lambda_s \in \Re^{m_s}$ and $\nu_s \in \Re^{n_s}$ are the dual variables for the second stage equality constraints and the bounds. The total number of variables is $n := n_0 + \sum_{s \in \mathcal{S}} n_s$ and the total number of equality constraints is $m := m_0 + \sum_{s\in\mathcal{S}} m_s$. If we denote $x^T := [x_0^T, x_1^T, ..., x_S^T]$, this problem is a general NLP problem. However, specific solvers can be developed to take advantage of the problem structures.

If we use interior point method to solve the problem (6.3), the KKT system has the following arrowhead form after reformulation

$$\begin{bmatrix} K_1 & & & & B_1 \\ & K_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & K_S & B_S \\ B_1^T & B_2^T & \dots & B_S^T & K_0 \end{bmatrix} \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_S \\ \Delta w_0 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_S \\ r_0 \end{bmatrix}, \tag{6.4}$$

Assuming that all K_s are of full rank, we can show with the Schur complement method that the solution of the Equation (3.17) is equivalent to that of the following system

$$\underbrace{(K_0 - \sum_{s\in\mathcal{S}} B_s^T K_s^{-1} B_s)}_{:=Z} \Delta w_0 = \underbrace{r_0 - \sum_{s\in\mathcal{S}} B_s^T K_s^{-1} r_s}_{:=r_Z} \tag{6.5a}$$

$$K_s \Delta w_s = r_s - B_s \Delta w_0, \quad \forall s \in \mathcal{S}. \tag{6.5b}$$

The system (6.5) can be solved with 3 steps. The first step is to form Z and r_Z by adding the contribution from each block. This step requires the factorizations of one sparse matrix K_1 of size $n_1 + 2n_0 + m_1 + m_0$ and $S - 1$ sparse matrix K_s of size $n_s+2n_0+m_s$. Besides a total of S factorizations of block matrix, this step also requires a total of $(S + 1)n_0$ backsolves. The second step is to solve the Equation (6.5a) to get direction of the first stage variables Δw_0. This step requires one factorization and one backsolve of the dense matrix Z. With Δw_0, the third step is to compute Δw_s from Equation (6.5b). This step requires a total of S backsolves of the block spase matrix.

One significant advantage of solving the system (3.19) is that both step 1 and step 3 can be easily parallelized. When n_0 is relatively small, and thus the cost of factorizing matrix Z in step 2 is negligible, the efficiency of the parallel implementation can be very close to 1. Another advantage of using the parallel Schur complement method on distributed architectures is that the memory requirement is much smaller for each node than solving the system (3.6). For the batch process of crystallization, if we discretize the control by 18 steps, the total number of first stage variables is 19, which is small enough that the explicit Schur complement method is still efficient.

6.3 Performance of Robust NMPC on Batch Crystalization

In this section, we evaluate the performance of the min-max NMPC with six uncertain parameters. We assume that k_b, b, k_{g_1}, g_1, k_{g_2} and g_2 follow uniform distributions on the interval $[3.494 \cdot 10^6 \quad 5.494 \cdot 10^6]$ #/cm^3 min, [2.02 2.06], [0.06326 0.08326] cm/min, [1.46 1.50], [0.5045 0.7045] cm/min, [1.72 1.76]. We also assume that the measurement noise corresponding to three measurements ML, AR and C follows truncated normal distributions on the interval $[-12 \quad 12]$ μm, $[-0.2 \quad 0.2]$, and $[-0.008 \quad 0.008]$ g/cm^3. The mean values of the original normal distribution are all zero, and the standard deviations are 6 μm, 0.1, and 0.004 g/cm^3, respectively. The setpoint s_4 we chose is AR_{set_4}=2.9 and ML_{set_4}=200μm. Nominal values of parameters are selected according to Table 5.1. The performance of a specific test scenario is still evaluated by the cost function Eq. (5.15). We use 50 scenarios generated from the uncertain parameter distribution to test the robust performance, and we will refer to these scenarios as test scenarios. We will use another set of scenarios in the NMPC optimization to determine the optimal control profile, and we will refer to these scenarios as model scenarios.

Table 6.1 shows the robust performance of different control strategies. "Ideal" is the NMPC/open-loop with accurate knowledge about the value of true parameter used. "Open Loop" is the open loop performance with nominal value. "NMPC" is the performance of NMPC with nominal value $kb = 4.494 \cdot 10^6$. It also uses

Table 6.1: The robust performance (value of *cost*) of different control strategies when six parameters have uncertainties.

	Nominal	Average	Standard Deviation	Worst Case
Ideal	2e-4	44	68	317
Open Loop	2e-4	287	289	1626
NMPC	2e-1	171	221	1321
NMPC P	2e-1	131	169	1061
Exact Min-max NMPC	42	117	128	573

MHE to estimated unknown state variables. "NMPC P" is similar to NMHE but uncertain parameters are estimated by MHE and provided to NMPC. "Exact min-max NMPC" is the performance of robust NMPC with the model scenarios exactly the same as the test scenarios. For this case, even if the NMPC knows exactly the value of uncertain parameters, it cannot achieve the setpoint for several test scenarios. The worst case performance for the ideal NMPC with exact parameters are 317. This is the lower bound of the worst case performance of all other control strategies. The robust performance of the NMPC with parameter updates is better than that of the NMPC without parameter updates. This is likely because the measurement noise is relatively small compared with the parameter uncertainties. In this case, the robust performance of the robust NMPC is much better than that of nominal MPC in terms of average, standard deviation and worst case *cost* evaluated by 50 test scenarios. However, the robust NMPC needs to sacrifice the nominal performance when the uncertain parameters are all at their nominal values. Compared with the reduction in worst case *cost*, the nominal *cost* is still small. Figure 6.1 shows that the optimal temperature profiles from the nominal NMPC and the robust NMPC are quite different.

One drawback of "Exact Min-max NMPC" is that it uses the same scenarios for testing the performance to optimize the control profile in the robust NMPC model.

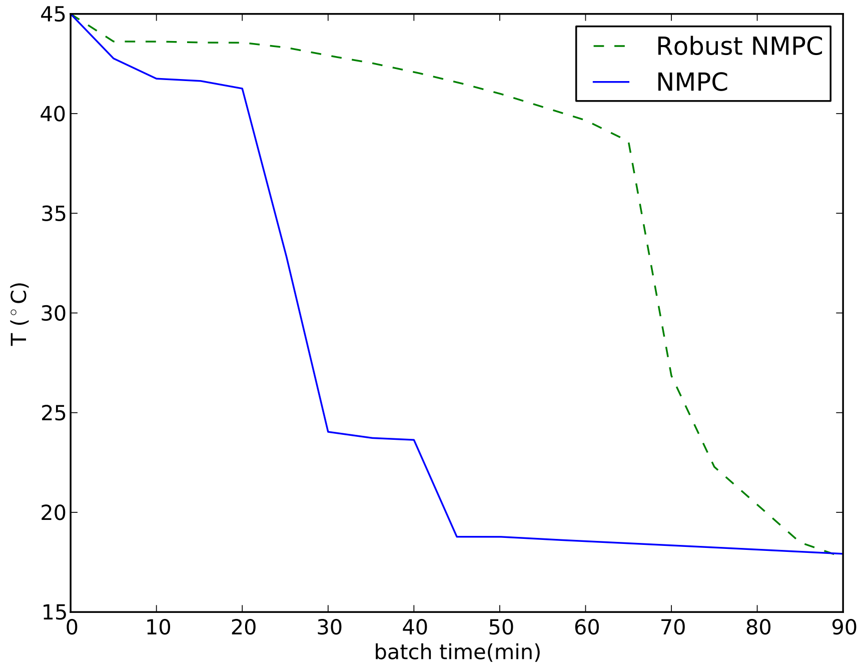


Figure 6.1.: Optimal temperature profile for nominal NMPC and robust NMPC.

Therefore we generate a different scenario set in the NMPC model from the uncertain parameter distributions. Table 6.2 shows the robust performance of min-max NMPC using different numbers of model scenarios. This table shows that increasing the number of model scenarios can in general improve the performance. However, many other factors (e.g. similarity of model scenarios and testing scenarios) also influence the robust performance.

The size of the problem solved in Table 6.2 with 150 model scenarios is very large. It has 651319 variables and 651300 constraints. Table 6.3 shows the solution time of solving 1 optimization problem at step $t = 0$. The total time is composed of both the time constructing the model and the time solving the NLP. Using the Schur complement method can not only solve the problem in parallel but also build the

Table 6.2: The robust performance of the robust NMPC using different numbers of scenarios evaluated using 50 simulations.

S	Nominal	Average	Standard Deviation	Worst Case
Exact	42	117	128	573
50	13	163	192	1128
100	41	116	136	701
150	33	124	144	808

model in parallel. It gains 11 times speedup on a computer with 15 cores compared with its own serial implementation. A solver using full factorization method like IPOPT takes more than 1 hour to solve the problem while parallel Schur complement solver takes less than 1 minute.

Table 6.3: The solution time of solving a robust optimization problem with 150 scenarios.

	# Processors	Full Factorization	Schur Complement Method	
		Time(s)	Time(s)	Speedup
Building Model	1	67.7	97.6	-
	2	-	52.6	1.9
	5	-	22	4.4
	10	-	12.1	8.0
	15	-	8.5	11.5
Solving NLP	1	≥ 3600	592	-
	2	-	313.9	1.9
	5	-	129.4	4.6
	10	-	71.7	8.3
	15	-	51.8	11.4

6.4 Performance of Robust NMPC with Bayesian Inference on Batch Crystalization

Uncertain parameters can be estimated using MHE. However, in the presence of significant noise and large uncertainties, point estimation results might not be accurate. Nevertheless, we can use Bayesian inference to update the posterior distributions of uncertainties and generate model scenarios used in the min-max NMPC according to the posterior distribution instead of prior distribution at each sampling instance.

Specifically, if we denote the uncertain parameter as p and measurements as y^{meas}. The posterior distribution is

$$f(p|y^{meas}) = \frac{f(y^{meas}|p)f(p)}{f(y^{meas})} \propto f(y^{meas}|p)f(p) \tag{6.6}$$

Where, $f(p)$ is the prior probability density before y^{meas} is observed, $f(y^{meas}|p)$ is the probability density of observing y^{meas} with a given p, and $f(y^{meas})$ is the probability density of observing y^{meas}. Since it is the same for all p, it can be viewed as a constant. For a given p, we can get a corresponding $y(p)$ from simulation. Therefore $f(y^{meas}|p)$ is equivalent to $f(y^{meas}|y(p))$ and can be computed according to the measurement error distribution. With these information, Markov chain Monte Carlo (MCMC) can be used to generate a set of scenarios.

One drawback of min-max NMPC is that it also takes into consideration of some uncertain scenarios that have very low probability. In our implementation, after S scenarios are generated, we first compute the relative probability of each scenario within the model scenario set by

$$Pr(p_s|y^{meas}) = \frac{f(y^{meas}|p_s)f(p_s)}{\sum_{s\in\mathcal{S}} f(y^{meas}|p_s)f(p_s)} \tag{6.7}$$

If the posterior distribution also follows a uniform distribution, the relative probability should be $1/S$ for each scenario. If the relative probability of p_s is smaller than $10^{-6}/S$, p_s is discarded and a new scenario is generated.

Table 6.4 illustrates that the performance of min-max NMPC with Bayesian inference with 50 model scenarios at each sampling instance is close to the ideal performance. Increasing the number of scenarios can improve the robust performance. The performance of Bayesian min-max NMPC with 12 scenarios is already close to that of conventional min-max NMPC with 50 exact scenarios.

Table 6.4: Robust Performance of min-max NMPC with Bayesian inference using different numbers of model scenarios evaluated using 50 simulations.

S	Nominal	Average	Standard Deviation	Worst Case
12	17.0	99	121	649
25	15.2	96	122	482
50	13.6	72	87	378

6.5 Concluding Remarks

In conclusion, robust NMPC not only ensures that the constraints are satisfied for all uncertain scenarios if each optimization can find a feasible solution, but also provides a reliable way to get moderate robust performance, especially when there are multiple uncertain parameters and the uncertainty is large. The performance of robust NMPC can be improved by generating model scenarios from posterior distribution from Bayesian inference.

7. SUMMARY

The demand for fast solution of nonlinear optimization problems, coupled with the emergence of new concurrent computing architectures, drives the need for parallel algorithms to solve challenging nonlinear programming (NLP) problems. The objective of this dissertation is to develop parallel algorithms to solve structured and unstructured large-scale nonlinear programming (NLP) problems. This chapter first summarizes our contributions and then makes suggestions for future work.

7.1 Thesis Summary and Contributions

Chapter 1 highlights the importance of solving large-scale NLP problems in parallel and gives an introduction of the parallel architectures and the current state-of-art in parallel NLP algorithms. The problems addressed by these algorithms can be classified into two categories: one is the general unstructured NLP problems and the other is structured NLP problems (such as stochastic programs). One algorithm for the first class of problems is discussed in Chapter 2 and two algorithms for the second class of problems are discussed in Chapters 3 and 4.

Chapter 2 proposes a parallel algorithm on the GPU for general NLP problems. The main contributions in Chapter 2 are:

- The first algorithm for solving large-scale unstructured constrained NLP problems using graphics processing units. The advantage of the augmented Lagrangian approach is that the KKT system is positive definite for convex problems, which enables us to solve the KKT system using a parallel PCG method on the GPU.

An overall speedup of 13-18 was obtained on six test problems from COPS test set. Three major algorithm optimizations were implemented in order to achieve these speedups.

- First, since each PCG iteration only requires a series of matrix-vector products with J^k, the PCG iterations were performed without explicitly forming J^k. Second, in order to ensure improved coalesced and aligned global memory access on the GPU, different matrix storage formats were utilized as appropriate. Lastly, we implemented problem specific code for parallel function and derivative evaluations on the GPU.

Chapter 3 describes a parallel Schur complement method for nonlinear stochastic programs. When the number of first stage variables is small, this approach has almost perfect efficiency. However, the performance of this approach quickly deteriorates as the number of first stage variables increases. This disadvantage is overcome by the algorithm proposed in Chapter 4. Chapter 4 presents the following contributions:

- The first parallel algorithm to solve stochastic programs within interior point framework not based on Schur complement method. The algorithm performs adaptive clustering of scenarios (inside-the-solver) based on their influence on the problem to form a preconditioner. The preconditioner is then used by an iterative solver to solve the KKT system.

The preconditioners can be implemented in sparse form and dramatically reduce computational time compared to full factorizations of the KKT system. The sparse form enables the solution of problems with large first-stage dimensionality that cannot be addressed with explicit Schur complement method. This parallel algorithm is used to solve a market clearing problem with a speed up factor of 42 compared to the full factorization method. Scenario compression rates of up to 94 % are shown to be possible.

The second half of this dissertation describes the application of nonlinear programming in pharmaceutical manufacturing. we look at a specific manufacturing unit and seeks to control the product quality in batch crystallization process. Chapter 5 presents the following contributions:

- We design and develop real-time feasible multi-objective optimization based NMPC-MHE formulations for batch crystallization processes to control the crystal size and shape distribution.

At each sampling instance, based on a nonlinear DAE model, an estimation problem estimates unknown states and parameters and an optimal control problem determines the optimal input profiles. Both DAE-constrained optimization problems are solved by discretizing the system using Radau collocation and optimizing the resulting algebraic nonlinear problem using Ipopt. The performance of this control strategy is analyzed in terms of setpoint change, system noise, and model/plant mismatch. For all cases, the performance of the NMPC-MHE is shown to provide better setpoint tracking than the open-loop optimal control strategy. Furthermore, the combined solution time for the MHE and the NMPC formulations is well within the sampling interval, allowing for real world application of the control strategy.

The quality of the NMPC approach depends on the accuracy of the underlying model. Despite the high fidelity of using the nonlinear models based on the first principles, there are still uncertainties associated with external and internal disturbances. To deal with the parameter uncertainties in the crystallization model, Chapter 6 presents the following contributions:

- We design and develop real-time feasible robust NMPC formulations for batch crystallization processes to minimize the deviation of the product quality from the setpoint in the worst case. The size of optimization problems solved online becomes too large to be solved by a serial solver, and the algorithm described in Chapter 3 is used to solve the robust NMPC problems.

Robust NMPC not only ensures the constraints are satisfied for all uncertain scenarios if each optimization can find a feasible solution, but also provides a consistent way to get moderate robust performance especially when there are multiple uncertain parameters and the uncertainties are large.

7.2 Future Work

The following are some recommendations for future work:

- The augmented Lagrangian approach as implemented in Chapter 2 is best for problems with few equality constraints. Future work will explore modifications to handle more equalities. For example, the augmented Lagrangian algorithm used by MINOS is better for problems with little degrees of freedom. In addition, we used a straightforward diagonal preconditioner, and other parallel capable preconditioners should be investigated. Finally, we manually implemented routines for parallel model evaluations, and this approach should be automated.

- The clustering preconditioning approach as implemented in Chapter 4 are designed for convex stochastic QP problems. We will investigate the performance of the preconditioner in a nonlinear programming setting, and we will investigate extensions to multi-stage stochastic programs.

- We will test the robust NMPC approach in Chapter 6 with more model scenarios and more cores. Also, although there are lots of statistical inference about stochastic programs minimizing the expected performance index, we need to investigate more about the statistical inference about the stochastic programs minimizing the worst case performance index.

LIST OF REFERENCES

LIST OF REFERENCES

Acevedo, D. and Z. K. Nagy (2014). Systematic classification of unseeded batch crystallization systems for achievable shape and size analysis. *Journal of Crystal Growth 394*, 97–105.

Acevedo, D., Y. Tandy, and Z. K. Nagy (2015). Multiobjective optimization of an unseeded batch cooling crystallizer for shape and size manipulation. *Industrial & Engineering Chemistry Research 54*(7), 2156–2166.

Agullo, E., J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov (2009, July). Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series 180*, 012037.

Åkesson, J., K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit (2010). Modeling and optimization with optimica and jmodelica. orglanguages and tools for solving large-scale dynamic optimization problems. *Computers & Chemical Engineering 34*(11), 1737–1749.

Alvarez, A. J. and A. S. Myerson (2010). Continuous plug flow crystallization of pharmaceutical compounds. *Crystal Growth & Design 10*(5), 2219–2228.

Amestoy, P. R., I. S. Duff, and J.-Y. L'Excellent (2000). Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer methods in applied mechanics and engineering 184*(2), 501–520.

Baskaran, M. M. and R. Bordawekar (2008). Optimizing sparse matrix-vector multiplication on gpus using compile-time and run-time strategies. *IBM Reserach Report, RC24704 (W0812-047).*

Bell, N. and M. Garland (2009). Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 18. ACM.

Bergamaschi, L., J. Gondzio, and G. Zilli (2004). Preconditioning indefinite systems in interior point methods for optimization. *Computational Optimization and Applications 28*(2), 149–171.

Biegler, L. T. (2007). An overview of simultaneous strategies for dynamic optimization. *Chemical Engineering and Processing: Process Intensification 46*(11), 1043–1053.

Biegler, L. T. (2010). *Nonlinear programming: concepts, algorithms, and applications to chemical processes*, Volume 10. SIAM.

Biegler, L. T., A. M. Cervantes, and A. Wächter (2002). Advances in simultaneous strategies for dynamic process optimization. *Chemical Engineering Science 57*(4), 575–593.

Birge, J. (1985). Aggregation bounds in stochastic linear programming. *Mathematical Programming 31*, 25–41.

Bishop, C. et al. (2006). *Pattern recognition and machine learning*, Volume 4. Springer, New York.

Braatz, R. D. (2002). Advanced control of crystallization processes. *Annual reviews in control 26*(1), 87–99.

Buatois, L., G. Caumon, and B. Levy (2009). Concurrent number cruncher: a gpu implementation of a general sparse linear solver. *International Journal of Parallel, Emergent and Distributed Systems 24*(3), 205–223.

Byrd, R. H., G. M. Chin, W. Neveitt, and J. Nocedal (2011). On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM Journal on Optimization 21*(3), 977–995.

Byrd, R. H., N. I. Gould, J. Nocedal, and R. A. Waltz (2003). An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Mathematical Programming 100*(1), 27–48.

Calafiore, G. C. and M. C. Campi (2006). The scenario approach to robust control design. *Automatic Control, IEEE Transactions on 51*(5), 742–753.

Cao, C., J. Dongarra, P. Du, M. Gates, P. Luszczek, and S. Tomov (2013). clmagma: High performance dense linear algebra with opencl. *University of Tennessee Computer Science Technical Report (Lawn 275).*

Cao, Y., C. Laird, and V. M. Zavala (2015). Clustering-based preconditioning for stochastic programs. *submited to Computational optimization and applications.*

Cao, Y., A. Seth, and C. Laird (2015). An augmented lagrangian interior-point approach for large-scale nlp problems on graphics processing units. *Computers & Chemical Engineering In Press.*

Casey, M. and S. Sen (2005). The scenario generation algorithm for multistage stochastic linear programming. *Mathematics of Operations Research 30*(3), 615–631.

Chiang, N. and A. Grothey (2012). Solving security constrained optimal power flow problems by a structure exploiting interior point method. *Optimization and Engineering*, 1–23.

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

Colombo, M., J. Gondzio, and A. Grothey (2011). A warm-start approach for large-scale stochastic linear programs. *Mathematical Programming 127*(2), 371–397.

Conn, A. R., N. I. Gould, and P. L. Toint (1988). Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation 50*(182), 399–430.

Corrigan, A., F. F. Camelli, R. Lhner, and J. Wallin (2011). Running unstructured grid based CFD solvers on modern graphics hardware. *International Journal for Numerical Methods in Fluids 66*, 221–229.

de Oliveira, W. L., C. Sagastizábal, D. Penna, M. Maceira, and J. M. Damázio (2010). Optimal scenario tree reduction for stochastic streamflows in power generation planning problems. *Optimization Methods and Software 25*(6), 917–936.

Dembo, R. S. and T. Steihaug (1983). Truncated-newton algorithms for large-scale unconstrained optimization. *Mathematical Programming 26*(2), 190–212.

Dick, C., J. Georgii, and R. Westermann (2011). A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simulation Modelling Practice and Theory 19*, 801–816.

Dolan, E. D., J. J. Moré, and T. S. Munson (2004). Benchmarking optimization software with cops 3.0. *Argonne National Laboratory Technical Report ANL/MCS-TM-273*.

Dollar, H. (2007). Constraint-style preconditioners for regularized saddle point problems. *SIAM Journal on Matrix Analysis and Applications 29*(2), 672–684.

Dollar, H. S., N. I. Gould, W. H. Schilders, and A. J. Wathen (2006). Implicit-factorization preconditioning and iterative solvers for regularized saddle-point systems. *SIAM Journal on Matrix Analysis and Applications 28*(1), 170–189.

Dupačová, J., N. Gröwe-Kuska, and W. Römisch (2003). Scenario reduction in stochastic programming. *Mathematical programming 95*(3), 493–511.

Elble, J. M., N. V. Sahinidis, and P. Vouzis (2010). Gpu computing with kaczmarzs and other iterative algorithms for linear systems. *Parallel computing 36*(5), 215–231.

Ferris, M. C. and T. S. Munson (2002). Interior-point methods for massive support vector machines. *SIAM Journal on Optimization 13*(3), 783–804.

Fletcher, R. and S. Leyffer (2002). Nonlinear programming without a penalty function. *Mathematical programming 91*(2), 239–269.

Forsgren, A., P. E. Gill, and J. D. Griffin (2007). Iterative solution of augmented systems arising in interior methods. *SIAM Journal on Optimization 18*(2), 666–690.

Forsgren, A., P. E. Gill, and M. H. Wright (2002). Interior methods for nonlinear optimization. *SIAM review 44*(4), 525–597.

Fujiwara, M., Z. K. Nagy, J. W. Chew, and R. D. Braatz (2005). First-principles and direct design approaches for the control of pharmaceutical crystallization. *Journal of Process Control 15*(5), 493–504.

Galoppo, N., N. K. Govindaraju, M. Henson, and D. Manocha (2005). Lu-gpu: Efficient algorithms for solving dense linear systems on graphics hardware. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, pp. 3. IEEE Computer Society.

Gay, D. M. and B. Kernighan (2002). Ampl: A modeling language for mathematical programming. *Duxbury Press/Brooks/Cole 2.*

Gill, P. E., W. Murray, and M. A. Saunders (2002). Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM journal on optimization 12*(4), 979–1006.

Golub, G. H. and C. F. Van Loan (2012). *Matrix computations*, Volume 3. JHU Press.

Gondzio, J. and A. Grothey (2003). Reoptimization with the primal-dual interior point method. *SIAM Journal on Optimization 13*, 842–864.

Gondzio, J. and A. Grothey (2009). Exploiting structure in parallel implementation of interior point methods for optimization. *Computational Management Science 6*(2), 135–160.

Gould, N. I., M. E. Hribar, and J. Nocedal (2001). On the solution of equality constrained quadratic programming problems arising in optimization. *SIAM Journal on Scientific Computing 23*(4), 1376–1395.

Gunawan, R., D. L. Ma, M. Fujiwara, and R. D. Braatz (2002). Identification of kinetic parameters in multidimensional crystallization processes. *International Journal of Modern Physics B 16*(01n02), 367–374.

Harris, M. (2007). Optimizing parallel reduction in cuda. *NVIDIA Developer Technology 6.*

Haseltine, E. L. and J. B. Rawlings (2005). Critical evaluation of extended kalman filtering and moving-horizon estimation. *Industrial & engineering chemistry research 44*(8), 2451–2460.

Heitsch, H. and W. Römisch (2009). Scenario tree reduction for multistage stochastic programs. *Computational Management Science 6*, 117–133.

Helfenstein, R. and J. Koko (2011). Parallel preconditioned conjugate gradient algorithm on gpu. *Journal of Computational and Applied Mathematics.*

Hogg, J. and J. Scott (2010). An indefinite sparse direct solver for large problems on multicore machines.

Huang, R., S. C. Patwardhan, and L. T. Biegler (2009). Multi-scenario-based robust nonlinear model predictive control with first principle models. *Computer Aided Chemical Engineering 27*, 1293–1298.

Huchette, J., M. Lubin, and C. Petra (2014). Parallel algebraic modeling for stochastic optimization. In *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages*, pp. 29–35. IEEE Press.

Hulburt, H. M. and S. Katz (1964). Some problems in particle technology: A statistical mechanical formulation. *Chemical Engineering Science 19*(8), 555–574.

Jiang, Z.-P. and Y. Wang (2001). Input-to-state stability for discrete-time nonlinear systems. *Automatica 37*(6), 857–869.

Jr., F. M., T. Szakly, R. Mszros, and I. Lagzi (2010). Air pollution modeling using a Graphics processing united with CUDA. *Computational Physics Communications 181*, 105–112.

Jung, J., D. OLeary, and A. Tits (2012). Adaptive constraint reduction for convex quadratic programming. *Computational Optimization and Applications 51*, 125–157.

Jung, J., D. P. Oleary, and A. L. Tits (2008). Adaptive constraint reduction for training support vector machines. *Electronic Transactions on Numerical Analysis 31*, 156–177.

Kang, J., Y. Cao, D. P. Word, and C. Laird (2014). An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition. *Computers & Chemical Engineering In Press*.

Krawezik, G. P. and G. Poole (2010). Accelerating the ansys direct sparse solver with gpus. *2010 Symposium on Application Accelerators in High Performance Computing (SAAHPC10)*.

Kumbhar, P. (2011). *Performance of PETSc GPU Implementation with Sparse Matrix Storage Schemes*. Ph. D. thesis, MSc Thesis, University of Edinburgh, Edinburgh: University of Edinburgh.

Latorre, J. M., S. Cerisola, and A. Ramos (2007). Clustering algorithms for scenario tree generation: Application to natural hydro inflows. *European Journal of Operational Research 181*(3), 1339 – 1353.

Li, R. and Y. Saad (2013). Gpu-accelerated preconditioned iterative linear solvers. *The Journal of Supercomputing 63*(2), 443–466.

Lin, C.-J. and J. J. Moré (1999). Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization 9*(4), 1100–1127.

Linderoth, J., A. Shapiro, and S. Wright (2006). The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research 142*(1), 215–241.

Lubin, M., C. Petra, and M. Anitescu (2012). The parallel solution of dense saddle-point linear systems arising in stochastic programming. *Optimization Methods and Software 27*(4-5), 845–864.

Lubin, M., C. G. Petra, M. Anitescu, and V. M. Zavala (2011). Scalable stochastic optimization of complex energy systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–10. IEEE.

Lucas, R. F., G. Wagenbreth, J. J. Tran, and D. M. Davis (2012). Multifrontal sparse matrix factorization on graphics processing units. *Information Sciences Institute, University of Southern California, Tech. Rep*.

Lukšan, L. and J. Vlček (1998). Indefinitely preconditioned inexact newton method for large sparse equality constrained nonlinear programming problems. *Numerical linear algebra with applications 5*(3), 219–247.

Ma, D. L., D. K. Tafti, and R. D. Braatz (2002). Optimal control and simulation of multidimensional crystallization processes. *Computers & Chemical Engineering 26*(7), 1103–1116.

Magni, L., G. De Nicolao, R. Scattolini, and F. Allgöwer (2003). Robust model predictive control for nonlinear discrete-time systems. *International Journal of Robust and Nonlinear Control 13*(3-4), 229–246.

Magni, L. and R. Scattolini (2007). Robustness and robust design of mpc for nonlinear discrete-time systems. In *Assessment and future directions of nonlinear model predictive control*, pp. 239–254. Springer.

Majumder, A. and Z. K. Nagy (2013). Prediction and control of crystal shape distribution in the presence of crystal growth modifiers. *Chemical Engineering Science 101*, 593–602.

Mayne, D. Q., J. B. Rawlings, C. V. Rao, and P. O. Scokaert (2000). Constrained model predictive control: Stability and optimality. *Automatica 36*(6), 789–814.

Mehrotra, S. (1992). On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization 2*, 575–601.

Mesbah, A., A. E. Huesman, H. J. Kramer, Z. K. Nagy, and P. M. Van den Hof (2011). Real-time control of a semi-industrial fed-batch evaporative crystallizer using different direct optimization strategies. *AIChE journal 57*(6), 1557–1569.

Mesbah, A., H. J. Kramer, A. E. Huesman, and P. M. Van den Hof (2009). A control oriented study on the numerical solution of the population balance equation for crystallization processes. *Chemical Engineering Science 64*(20), 4262–4277.

Mesbah, A., Z. Nagy, A. Huesman, H. Kramer, and P. Van den Hof (2012). Real-time control of industrial batch crystallization processes using a population balance modeling framework. *IEEE Trans. Control Syst. Technol 20*(5), 1188–1201.

Murtagh, B. A. and M. A. Saunders (1982). *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*. Springer.

Nagy, Z. K. and R. D. Braatz (2003). Robust nonlinear model predictive control of batch processes. *AIChE Journal 49*(7), 1776–1786.

Nagy, Z. K., G. Fevotte, H. Kramer, and L. L. Simon (2013). Recent advances in the monitoring, modelling and control of crystallization systems. *Chemical Engineering Research and Design 91*(10), 1903–1922.

Naumov, M. (2011). Incomplete-lu and cholesky preconditioned iterative methods using cusparse and cublas. *Nvidia white paper*.

Nocedal, J. and S. Wright (1999). *Numerical Optimization*. New York, NY: Springer.

Nocedal, J. and S. J. Wright (2006). *Numerical optimization*. Springer Science+ Business Media.

NVIDIA (2011). CUDA Programming Guide, Version 4.1.

NVIDIA (2012). CUDA C Best Practices Guide, Version 4.1.

Patience, D. B. and J. B. Rawlings (2001). Particle-shape monitoring and control in crystallization processes. *American Institute of Chemical Engineers. AIChE Journal 47*(9), 2125.

Perugia, I. and V. Simoncini (2000). Block-diagonal and indefinite symmetric preconditioners for mixed finite element formulations. *Numerical linear algebra with applications 7*(7-8), 585–616.

Petra, C. and M. Anitescu (2012). A preconditioning technique for Schur complement systems arising in stochastic optimization. *Computational Optimization and Applications 52*, 315–344.

Prasad, V., M. Schley, L. P. Russo, and B. W. Bequette (2002). Product property and production rate control of styrene polymerization. *Journal of Process Control 12*(3), 353–372.

Pritchard, G., G. Zakeri, and A. Philpott (2010). A single-settlement, energy-only electric power market for unpredictable and intermittent participants. *Operations Research 58*(4-part-2), 1210–1219.

Puel, F., G. Févotte, and J. Klein (2003). Simulation and analysis of industrial crystallization processes through multidimensional population balance equations. part 1: a resolution algorithm based on the method of classes. *Chemical Engineering Science 58*(16), 3715–3727.

Qamar, S., S. Mukhtar, A. Seidel-Morgenstern, and M. P. Elsner (2009). An efficient numerical technique for solving one-dimensional batch crystallization models with size-dependent growth rates. *Chemical Engineering Science 64*(16), 3659–3667.

Qin, S. J. and T. A. Badgwell (2003). A survey of industrial model predictive control technology. *Control engineering practice 11*(7), 733–764.

Ramkrishna, D. (2000). *Population balances: Theory and applications to particulate systems in engineering.* Academic press.

Rao, C. V., J. B. Rawlings, and D. Q. Mayne (2003). Constrained state estimation for nonlinear discrete-time systems: Stability and moving horizon approximations. *Automatic Control, IEEE Transactions on 48*(2), 246–258.

Rawlings, J. B. (2000). Tutorial overview of model predictive control. *Control Systems, IEEE 20*(3), 38–52.

Rawlings, J. B. and B. R. Bakshi (2006). Particle filtering and moving horizon estimation. *Computers & chemical engineering 30*(10), 1529–1541.

Roman, G., Joldes, and K. M. Adam Wittek (2010). Real-time nonlinear finite element computations on GPU Application to neurosurgical simulation. *Computer Methods in Applied Mechanics and Engineering 199*, 3305–3314.

Schenk, O. and K. Gärtner (2004). Solving unsymmetric sparse systems of linear equations with pardiso. *Future Generation Computer Systems 20*(3), 475–487.

Scokaert, P. and D. Mayne (1998). Min-max feedback model predictive control for constrained linear systems. *Automatic Control, IEEE Transactions on 43*(8), 1136–1142.

Shapiro, A., D. Dentcheva, et al. (2014). *Lectures on stochastic programming: modeling and theory*, Volume 16. SIAM.

Shetty, C. M. and R. W. Taylor (1987). Solving large-scale linear programs by aggregation. *Computers & Operations Research 14*(5), 385 – 393.

Szyld, D. B. and J. A. Vogel (2001). Fqmr: A flexible quasi-minimal residual method with inexact preconditioning. *SIAM Journal on Scientific Computing 23*(2), 363–380.

Togkalidou, T., M. Fujiwara, S. Patel, and R. D. Braatz (2000). A robust chemometrics approach to inferential estimation of supersaturation. In *American Control Conference, 2000. Proceedings of the 2000*, Volume 3, pp. 1732–1736. IEEE.

Togkalidou, T., M. Fujiwara, S. Patel, and R. D. Braatz (2001). Solute concentration prediction using chemometrics and atr-ftir spectroscopy. *Journal of Crystal Growth 231*(4), 534–543.

Tomov, S., R. Nath, H. Ltaief, and J. Dongarra (2010, April). Dense linear algebra solvers for multicore with GPU accelerators. *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 1–8.

Toolkit, C. (2011). 4.0 cublas library. *NVIDIA Corporation*.

Vanderbei, R. J. and D. F. Shanno (1999). An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications 13*(1-3), 231–252.

Vouzis, P. D. and N. V. Sahinidis (2011). Gpu-blast: using graphics processors to accelerate protein sequence alignment. *Bioinformatics 27*(2), 182–188.

Wächter, A. (2002). *An interior point algorithm for large-scale nonlinear optimization with applications in process engineering.* Ph. D. thesis, PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Wächter, A. and L. T. Biegler (2005). Line search filter methods for nonlinear programming: Local convergence. *SIAM Journal on Optimization 16*(1), 32–48.

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

Waltz, R. A., J. L. Morales, J. Nocedal, and D. Orban (2006). An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming 107*(3), 391–408.

Wan, J., X. Z. Wang, and C. Y. Ma (2009). Particle shape manipulation and optimization in cooling crystallization involving multiple crystal morphological forms. *AIChE journal 55*(8), 2049–2061.

Wang, X., J. C. De Anda, and K. Roberts (2007). Real-time measurement of the growth rates of individual crystal facets using imaging and image analysis: a feasibility study on needle-shaped crystals of l-glutamic acid. *Chemical Engineering Research and Design 85*(7), 921–927.

Watson, J.-P., D. L. Woodruff, and W. E. Hart (2012). Pysp: modeling and solving stochastic programs in python. *Mathematical Programming Computation 4*(2), 109–149.

Yeralan, S. N., T. Davis, and S. Ranka (2013). Sparse QR factorization on GPU architectures. Technical report, Technical report, University of Florida (November 2013).

Zavala, V. M., A. Botterud, E. M. Constantinescu, and J. Wang (2010). Computational and economic limitations of dispatch operations in the next-generation power grid. *IEEE Conference on Innovative Technologies for and Efficient and Reliable Power Supply*.

Zavala, V. M., E. M. Constantinescu, T. Krause, and M. Anitescu (2009). On-line economic optimization of energy systems using weather forecast information. *Journal of Process Control 19*(10), 1725–1736.

Zavala, V. M., C. D. Laird, and L. T. Biegler (2008). Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chemical Engineering Science 63*(19), 4834 – 4845.

Zhang, Y., P. Vouzis, and N. V. Sahinidis (2011). Gpu simulations for risk assessment in co 2 geologic sequestration. *Computers & Chemical Engineering 35*(8), 1631–1644.

Zipkin, P. (1980). Bounds for row-aggregation in linear programming. *Operations Research 28*(4), 903–916.

APPENDICES

A. DETAILED PERFORMANCE OF DIFFERENT CONTROL STRATEGIES FOR 50 TEST SCENARIOS

Table A.1: The value of uncertain parameters in 50 tests.

Scenario No.	k_b (10^6/cm^3 min)	b	k_{g_1} (cm/min)	g_1	k_{g_2} (cm/min)	g_2
1	3.99	2.05	0.075	1.48	0.68	1.74
2	4.69	2.05	0.076	1.50	0.68	1.73
3	4.21	2.04	0.082	1.48	0.61	1.75
4	4.25	2.05	0.066	1.49	0.53	1.73
5	3.54	2.05	0.075	1.47	0.53	1.75
6	3.97	2.02	0.077	1.48	0.67	1.74
7	4.27	2.05	0.079	1.50	0.69	1.75
8	4.86	2.05	0.066	1.47	0.70	1.74
9	5.17	2.05	0.075	1.48	0.56	1.74
10	5.02	2.05	0.077	1.48	0.69	1.72
11	3.94	2.04	0.068	1.46	0.62	1.73
12	3.62	2.05	0.077	1.47	0.66	1.73
13	5.05	2.03	0.070	1.49	0.61	1.76
14	4.94	2.05	0.064	1.48	0.57	1.73
15	3.88	2.04	0.081	1.48	0.69	1.74
16	4.52	2.05	0.078	1.50	0.66	1.76
17	3.63	2.04	0.079	1.50	0.57	1.74
18	4.28	2.02	0.074	1.50	0.69	1.74
19	5.46	2.02	0.073	1.48	0.64	1.73
20	4.10	2.02	0.077	1.48	0.62	1.73

21	5.06	2.04	0.082	1.47	0.57	1.74
22	4.96	2.05	0.078	1.47	0.52	1.73
23	5.17	2.06	0.071	1.46	0.59	1.75
24	3.74	2.03	0.077	1.48	0.56	1.73
25	5.23	2.03	0.064	1.49	0.60	1.76
26	4.62	2.02	0.076	1.47	0.54	1.72
27	5.17	2.03	0.070	1.48	0.67	1.73
28	4.97	2.04	0.066	1.46	0.66	1.73
29	5.07	2.06	0.072	1.49	0.54	1.73
30	4.86	2.02	0.081	1.49	0.61	1.75
31	5.32	2.06	0.067	1.49	0.64	1.75
32	3.94	2.04	0.063	1.47	0.53	1.75
33	4.64	2.03	0.083	1.50	0.59	1.73
34	4.75	2.06	0.071	1.48	0.66	1.74
35	3.52	2.03	0.070	1.48	0.54	1.72
36	4.02	2.05	0.070	1.46	0.62	1.72
37	4.95	2.03	0.072	1.49	0.60	1.73
38	5.08	2.05	0.065	1.47	0.56	1.76
39	4.14	2.03	0.082	1.47	0.60	1.72
40	4.05	2.04	0.076	1.47	0.53	1.73
41	3.74	2.04	0.080	1.50	0.68	1.74
42	4.31	2.03	0.066	1.48	0.54	1.74
43	5.40	2.03	0.073	1.50	0.57	1.74
44	3.97	2.03	0.068	1.47	0.66	1.75
45	3.51	2.05	0.069	1.46	0.60	1.72
46	3.53	2.05	0.068	1.46	0.68	1.73
47	4.81	2.03	0.067	1.46	0.64	1.73
48	4.86	2.03	0.065	1.47	0.69	1.74
49	4.90	2.05	0.074	1.50	0.64	1.73

50	4.90	2.04	0.078	1.50	0.60	1.75

Table A.2: Performance (value of *cost*) of Ideal control strategy when six parameters have uncertainty.

Scenario No.	AR	ML	*cost*
1	3.38	209.68	116.48
2	3.76	211.34	203.27
3	3.04	198.24	5.07
4	2.92	199.98	0.03
5	3.15	196.69	17.30
6	2.90	200.02	0.00
7	3.10	200.73	4.66
8	3.49	202.75	42.35
9	3.26	191.91	78.68
10	3.20	200.90	9.52
11	3.00	200.38	1.16
12	3.31	204.08	33.39
13	3.22	196.20	24.67
14	3.03	199.96	1.72
15	3.11	201.75	7.58
16	2.90	199.99	0.00
17	2.90	200.02	0.00
18	3.64	205.10	80.90
19	3.01	199.75	1.32
20	2.90	200.00	5.29E-07
21	3.37	186.31	209.14
22	3.34	184.90	247.13
23	3.31	194.97	42.10

24	2.90	199.99	0.00
25	3.19	197.46	14.62
26	3.32	192.69	71.45
27	3.52	202.19	43.56
28	3.17	200.71	7.77
29	2.98	194.29	33.28
30	3.30	194.73	43.52
31	3.23	200.29	11.14
32	2.97	199.06	1.40
33	2.98	196.91	10.15
34	3.24	201.04	12.32
35	2.90	200.01	0.00
36	3.46	207.39	85.41
37	3.02	199.95	1.33
38	3.43	193.49	70.02
39	2.90	199.98	0.00
40	3.13	195.27	27.53
41	3.35	205.07	45.86
42	2.91	199.95	0.01
43	3.17	192.60	62.11
44	3.00	200.29	1.02
45	3.26	202.33	18.41
46	3.70	215.93	317.49
47	3.01	199.96	1.26
48	3.62	203.64	65.67
49	3.67	207.02	109.03
50	3.12	196.44	17.67

Table A.3: Performance (value of *cost*) of open loop control strategy when six parameters have uncertainty.

Scenario No.	AR	ML	*cost*
1	3.11	220.12	409.38
2	3.43	223.47	578.86
3	2.47	188.95	140.12
4	3.08	203.37	14.61
5	2.30	185.40	249.55
6	3.04	212.13	149.04
7	3.18	217.82	325.75
8	3.54	222.27	536.37
9	2.66	184.60	243.13
10	3.28	218.59	359.74
11	3.12	216.64	281.43
12	3.10	226.80	722.11
13	2.90	188.88	123.65
14	3.24	206.46	53.26
15	2.98	218.71	350.85
16	2.94	203.34	11.30
17	2.81	205.02	26.11
18	3.57	226.30	736.57
19	3.17	198.70	8.80
20	2.91	204.26	18.15
21	2.34	176.08	603.31
22	2.35	176.77	570.03
23	2.58	184.65	245.75
24	2.61	196.36	21.37
25	3.16	193.17	53.33

26	2.60	184.63	245.32
27	3.60	220.23	457.78
28	3.36	215.01	246.40
29	2.80	189.93	102.31
30	2.67	185.38	218.93
31	3.46	211.28	158.34
32	2.77	193.26	47.09
33	2.75	192.18	63.24
34	3.33	218.29	352.72
35	2.95	207.08	50.39
36	3.13	221.09	450.25
37	3.20	203.26	19.71
38	2.71	182.90	296.28
39	2.57	195.00	36.12
40	2.47	187.19	182.62
41	3.23	227.80	783.46
42	2.97	196.86	10.41
43	2.96	187.66	152.69
44	3.19	214.41	215.97
45	3.11	224.03	581.81
46	3.43	239.98	1626.01
47	3.24	209.14	95.21
48	3.66	223.28	599.42
49	3.45	219.88	425.45
50	2.79	189.69	107.60

Table A.4: Performance (value of *cost*) of NMPC without parameter updates when six parameters have uncertainty.

Scenario No.	AR	ML	*cost*
1	3.47	215.29	266.82
2	3.77	217.06	366.61
3	2.76	194.10	36.70
4	3.06	200.44	2.71
5	2.47	190.14	115.59
6	3.25	207.68	71.23
7	3.52	212.71	199.27
8	3.71	216.40	334.15
9	2.94	188.53	131.74
10	3.51	213.10	208.20
11	3.34	210.95	138.79
12	3.56	220.80	476.30
13	3.06	193.08	50.56
14	3.24	201.35	13.39
15	3.31	214.39	224.02
16	2.96	199.20	0.94
17	2.89	201.81	3.30
18	3.85	220.45	508.54
19	3.12	199.46	5.22
20	2.91	203.02	9.11
21	2.67	179.84	411.98
22	2.67	179.89	409.69
23	2.91	189.82	103.56
24	2.86	199.82	0.23

25	3.11	195.98	20.48
26	2.95	188.22	139.08
27	3.72	215.04	293.96
28	3.64	209.92	153.79
29	3.03	193.26	47.09
30	2.98	189.85	103.58
31	3.39	206.21	62.85
32	2.84	195.06	24.80
33	2.93	195.97	16.29
34	3.56	212.35	196.78
35	3.12	202.56	11.30
36	3.53	214.81	259.54
37	3.15	200.42	6.67
38	3.01	188.28	138.56
39	2.82	199.29	1.13
40	2.67	191.45	78.24
41	3.70	221.55	529.09
42	2.99	198.50	3.04
43	3.19	191.38	82.53
44	3.58	210.61	159.56
45	3.40	217.48	330.96
46	4.03	234.53	1321.35
47	3.21	204.19	27.19
48	3.70	216.78	346.43
49	3.52	213.45	219.95
50	3.02	193.74	40.64

Table A.5: Performance (value of *cost*) of NMPC with parameter updates when six parameters have uncertainty.

Scenario No.	AR	ML	*cost*
1	3.32	212.41	171.85
2	3.69	211.48	194.81
3	3.49	200.04	34.73
4	3.17	200.28	7.37
5	3.21	195.46	30.13
6	3.53	210.52	150.89
7	3.32	207.47	73.18
8	3.69	211.74	199.77
9	3.27	190.72	100.01
10	3.64	208.66	129.32
11	3.35	207.49	76.08
12	3.43	215.00	253.38
13	2.90	191.59	70.75
14	3.28	201.56	17.02
15	3.34	211.90	160.91
16	3.28	201.05	15.41
17	3.75	204.12	89.94
18	3.94	218.13	436.22
19	3.05	199.29	2.86
20	2.95	203.06	9.60
21	3.06	183.39	278.54
22	3.06	182.36	313.83
23	3.51	194.89	63.34
24	3.09	198.43	5.95

25	3.06	195.16	25.96
26	3.50	191.72	104.78
27	3.90	210.11	202.70
28	3.90	207.07	150.38
29	3.09	193.75	42.82
30	3.63	195.21	76.07
31	3.37	204.87	46.06
32	2.92	194.88	26.23
33	3.44	197.50	35.36
34	3.68	209.39	149.69
35	3.32	201.47	19.54
36	3.56	213.96	238.21
37	3.10	200.19	3.87
38	3.46	192.19	92.83
39	2.96	200.19	0.44
40	3.18	194.18	41.83
41	3.52	215.52	278.87
42	3.11	198.78	6.02
43	3.16	190.66	94.15
44	3.23	206.71	56.27
45	3.45	217.84	348.22
46	3.73	231.50	1061.13
47	3.62	202.45	57.45
48	3.76	214.99	297.79
49	3.89	209.64	191.56
50	3.10	194.98	29.15

Table A.6: Performance (value of *cost*) of Exact Min-max NMPC when six parameters have uncertainty.

Scenario No.	AR	ML	*cost*
1	3.09	207.39	58.34
2	3.34	211.66	155.80
3	2.41	190.04	123.33
4	2.94	198.92	1.37
5	2.26	187.22	204.59
6	2.95	202.27	5.47
7	3.14	205.35	34.25
8	3.34	211.26	146.27
9	2.63	185.87	206.91
10	3.22	206.61	53.95
11	3.00	206.50	43.37
12	3.11	213.45	185.39
13	2.79	188.89	124.53
14	3.10	197.03	12.65
15	2.95	208.27	68.60
16	2.78	196.51	13.57
17	2.69	200.50	4.68
18	3.41	215.50	266.60
19	3.01	197.15	9.22
20	2.76	201.16	3.23
21	2.43	177.91	509.92
22	2.40	178.39	492.04
23	2.53	185.67	218.83
24	2.54	196.67	24.18
25	3.01	192.05	64.46

26	2.58	186.05	204.83
27	3.42	210.25	132.04
28	3.22	205.12	36.15
29	2.78	191.15	79.77
30	2.58	186.20	200.60
31	3.29	201.46	16.98
32	2.65	191.83	73.09
33	2.66	193.34	50.06
34	3.17	208.30	76.38
35	2.81	200.88	1.57
36	3.14	207.55	62.92
37	3.05	198.71	3.83
38	2.62	183.59	276.92
39	2.50	196.27	29.95
40	2.41	188.86	148.64
41	3.30	213.40	195.89
42	2.88	196.11	15.15
43	2.91	188.65	128.74
44	3.13	202.83	13.33
45	3.09	211.49	135.57
46	3.61	222.85	573.26
47	3.06	199.88	2.48
48	3.45	212.47	185.66
49	3.30	209.99	115.74
50	2.74	190.69	89.16

Table A.7: Performance (value of *cost*) of Bayesian min-max NMPC using 50 training scenarios when six parameters have uncertainty.

Scenario No.	AR	ML	*cost*
1	3.27	208.89	92.68
2	3.56	203.04	52.54
3	2.97	196.89	10.19
4	3.15	199.71	6.39
5	3.19	195.86	25.80
6	3.08	206.89	50.62
7	3.07	205.26	30.35
8	3.56	203.34	54.64
9	3.16	190.61	95.09
10	3.29	204.01	31.22
11	2.98	206.27	39.90
12	3.39	215.97	279.15
13	3.10	194.70	32.17
14	3.10	198.90	5.11
15	3.24	213.68	198.76
16	2.86	198.26	3.16
17	3.06	202.20	7.31
18	3.66	207.32	111.39
19	3.04	196.96	11.22
20	3.02	202.01	5.53
21	3.12	184.56	242.95
22	3.02	182.97	291.39
23	3.19	193.15	55.55
24	3.07	197.58	8.82

25	3.10	190.27	98.72
26	3.12	191.05	85.11
27	3.60	201.69	51.93
28	3.49	200.73	35.30
29	3.29	194.83	41.52
30	3.14	192.98	55.15
31	3.39	197.85	28.47
32	3.34	194.62	48.10
33	3.07	197.28	10.13
34	3.21	204.98	34.35
35	3.21	204.31	28.15
36	3.33	207.82	80.07
37	3.11	198.52	6.77
38	3.30	191.69	84.91
39	2.99	199.30	1.29
40	3.10	193.91	41.17
41	3.32	213.45	198.92
42	2.97	197.15	8.64
43	3.03	190.88	84.71
44	3.03	201.30	3.36
45	3.29	216.62	291.58
46	3.63	218.05	378.55
47	3.08	198.90	4.49
48	3.65	204.65	77.48
49	3.48	201.47	35.37
50	2.88	193.33	44.49

VITA

VITA

Yankai Cao was born in Ningbo, China. He received his bachelor's degree in Biological Engineering from Zhejiang University. In August 2010, he started graduate study at Texas A& M University, College Station and joined Dr. Carl Laird Group, where his research focuses on parallel algorithms for unstructured NLP problems and stochastic programs and their applications in pharmaceutical manufacturing. Yankai transferred to Purdue University with his advisor in January 2014. During the graduate study, Yankai did several internships - two at Argonne National Lab, one at United Airlines and one at Air Products. After graduation, he will work as a research associate at University of WisconsinMadison.