RIGOROUS METHODS FOR DYNAMIC OPTIMIZATION

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Yao Zhao

__________________________________________
Mark A. Stadtherr, Director

Graduate Program in Chemical and Biomolecular Engineering
Notre Dame, Indiana
April 2013
Abstract

by

Yao Zhao

Interval analysis and related verified solution methods provide mathematically and computationally guaranteed solution enclosures that bound all the possible solutions of dynamic systems with uncertainties. This provides a foundation for rigorous deterministic global optimization and continuous satisfaction of path constraints. Based on this methodology, we present here new algorithms for two types of problem: dynamic global optimization (DGO) subject to inequality path constraints (IPC) and robust design and global optimization with uncertain parameters.

The DGO method is based on techniques developed for the verified solution of parametric systems of ordinary differential equations. These techniques provide rigorous interval bounds on the state variables, and thus on the constraints and objective function in the DGO problem. These techniques also provide explicit analytic representations (Taylor models) of these bounds in terms of the decision variables. This facilitates the use of constraint propagation techniques that can greatly reduce the domain to be searched for the global optimum. Since IPCs are often related to safety concerns, we adopt a conservative approach to constraint satisfaction. Through this approach, the search for the global optimum is restricted to a space in which continuous satisfaction of the IPCs is rigorously guaranteed, and an $\epsilon$-global optimum within this space is determined.
We then extend the DGO method and adopt a bi-level branch and reduce structure to address robust design and global optimization with uncertain parameters. Within the framework, we implement a heuristic subregion management procedure to achieve more efficient decisions. This approach provides guaranteed-feasible decision variable regions for robust design problems and locates an $\epsilon$-global optimum over a confirmed feasible region for robust optimization problems.

Solving dynamic optimization problems with a rigorous approach can be computationally challenging. Therefore, another major goal here is to discuss how to tackle this challenge with parallel programming techniques. A general dynamic load balancing framework is designed, implemented using Message Passing Interface (MPI) and Posix multi-threads, and tested on a multi-core server. The effects of virtual networks and communication schemes are discussed.

Examples and comparative results for several problems are presented to demonstrate the potential of the methods presented and their computational performance.
To my parents, Zheng and Lu
CONTENTS

FIGURES .................................................................................. vi
TABLES .................................................................................. viii
ACKNOWLEDGMENTS ............................................................. x

CHAPTER 1: INTRODUCTION ....................................................... 1
1.1 Background ........................................................................ 1
1.2 Dynamic Global Optimization (DGO) Subject to Inequality Path
   Constraints (IPCs) ............................................................... 2
1.3 Robust Design and Global Optimization for Dynamic Systems .... 4
1.4 Parallel Computation ......................................................... 5
1.5 Outline ............................................................................. 7

CHAPTER 2: BACKGROUND ......................................................... 8
2.1 Interval Analysis ............................................................... 8
2.2 Taylor Models ................................................................. 13
2.3 Constraint Propagation with Taylor Models ......................... 17
2.4 Bounding State Variables ............................................... 21
   2.4.1 Phase One ........................................................... 22
   2.4.2 Phase Two ........................................................... 24
   2.4.3 Performance ......................................................... 24

CHAPTER 3: RIGOROUS GLOBAL OPTIMIZATION FOR DYNAMIC
   SYSTEMS SUBJECT TO INEQUALITY PATH CONSTRAINTS ........ 27
3.1 Introduction ................................................................. 27
3.2 Problem Statement ....................................................... 31
3.3 Global Optimization Algorithm ....................................... 34
   3.3.1 Initialization ......................................................... 35
   3.3.2 Rigorous IPC Test ............................................... 37
   3.3.3 Objective Test .................................................... 41
3.3.4 Overall Optimization Algorithm ........................................ 43
3.3.5 Tuning Procedure ......................................................... 46
3.4 Computational Studies ....................................................... 53
  3.4.1 Example 3.1: Second-order Exothermic Reaction in an Isother-
  mal Semi-batch Reactor ...................................................... 53
  3.4.2 Example 3.2: Exothermic Series Reaction in a Nonisother-
  mal Semi-batch Reactor ...................................................... 61
  3.4.3 Example 3.3: Parallel Reactions in an Isothermal Semi-
  batch Reactor ............................................................... 67

CHAPTER 4: ROBUST DESIGN AND GLOBAL OPTIMIZATION FOR
NONLINEAR DYNAMIC SYSTEMS ........................................... 73
  4.1 Introduction ................................................................. 73
  4.2 Problem Statement ......................................................... 75
  4.3 Solution Procedure ........................................................ 77
    4.3.1 Initialization .......................................................... 77
    4.3.2 Domain Elimination Rules .......................................... 78
    4.3.3 Feasibility Test ....................................................... 79
    4.3.4 Objective Test ........................................................ 80
    4.3.5 Sublist Management .................................................. 81
    4.3.6 Update Incumbent Solution and Sublists ......................... 83
    4.3.7 Bisection .............................................................. 83
    4.3.8 Overall Optimization Algorithm ................................... 84
  4.4 Computational Studies ..................................................... 85
    4.4.1 Example 4.1: Parameter Estimation ............................... 85
    4.4.2 Example 4.2: Batch Reactor Design ............................... 87
    4.4.3 Example 4.3: Saddle Point ........................................ 90
    4.4.4 Example 4.4: Haldane Law Bioreactor ............................ 92

CHAPTER 5: DYNAMIC LOAD BALANCING ..................................... 96
  5.1 Introduction ................................................................. 96
  5.2 Background ................................................................. 96
    5.2.1 Parallel Computation ............................................... 96
    5.2.2 Branch and Reduce Framework .................................... 97
    5.2.3 Parallelism Identification .......................................... 97
    5.2.4 Multi-process DLB ................................................... 99
    5.2.5 Multi-thread DLB .................................................... 99
  5.3 General DLB Workflow .................................................... 101
    5.3.1 Workload Initialization .............................................. 101
    5.3.2 Virtual Network Construction ..................................... 101
    5.3.3 Working Statues Exchange ......................................... 104
    5.3.4 Workload Placement ................................................ 105
FIGURES

2.1 Example of the wrapping effect ........................................ 13
3.1 Overall algorithm flowchart ........................................... 52
4.1 Parameter region subpaving plot of the batch reactor design example 89
4.2 Objective function value surface of example 4.3 ................. 92
5.1 All-to-all virtual network .............................................. 102
5.2 Worker manager virtual network ..................................... 103
5.3 1-D ring virtual network ............................................. 103
5.4 2-D mesh virtual network ............................................ 104
5.5 Illustration of SWS scheme ........................................... 110
5.6 Illustration of ADLB scheme ........................................ 112
5.7 Illustration of multi-threads access shared data .................... 114
5.8 Illustration of Mutex ................................................ 116
5.9 Comparison of load balancing schemes: Speedup vs. Number of processes ...................................................... 122
5.10 Comparison of load balancing schemes: Efficiency vs. Number of processes ...................................................... 122
5.11 Maximum and minimum number of work units tested on a single node out of a 16 nodes network ................................. 123
5.12 A distribution of elapsed time of three DLB schemes ............. 123
5.13 Comparison of virtual networks: Speedup vs. Number of processes ...................................................... 125
5.14 Comparison of virtual networks: Efficiency vs. Number of processes ...................................................... 125
5.15 Speedup achieved by using ADLB 2-D scheme on worst-case scenario ...................................................... 128
5.16 Speedup achieved by using ADLB 2-D scheme with different Taylor polynomial order ...................................................... 129
5.17 Parallel efficiency achieved by using ADLB 2-D scheme with different Taylor polynomial order ...................................................... 129
5.18 Comparison of load balancing schemes: Speedup vs. Number of threads ........................................ 130
5.19 Comparison of load balancing schemes: Efficiency vs. Number of threads ........................................ 131
5.20 Maximum and minimum number of work units tested on a single node out of a 16 nodes network .................. 131
5.21 A distribution of elapsed time of Pthreads implementation on two virtual networks .................................. 131
5.22 Comparison of MPI and Pthread: Wall time vs. Number of nodes .................................................. 132
5.23 Comparison of MPI and Pthread: Speedup vs. Number of nodes .................................................. 133
5.24 Comparison of MPI and Pthread: Efficiency vs. Number of nodes ........................................ 133
TABLES

3.1 MODEL PARAMETERS FOR EXAMPLE 1 ........................................ 55
3.2 RESULTS FOR EXAMPLE 3.1 WITH SAFETY CONSTRAINT ONLY AND INITIALIZATION OF $\phi$ WITH RANDOM SAMPLING 57
3.3 RESULTS FOR EXAMPLE 3.1 WITH SAFETY CONSTRAINT ONLY AND INITIALIZATION OF $\phi$ WITH ADJUSTED DOTCVP RESULTS ........................................... 58
3.4 RESULTS FOR EXAMPLE 3.1 WITH BOTH SAFETY AND VOLUME CONSTRAINTS AND INITIALIZATION OF $\phi$ WITH RANDOM SAMPLING ........................................... 59
3.5 RESULTS FOR EXAMPLE 3.1 WITH BOTH SAFETY AND VOLUME CONSTRAINTS AND INITIALIZATION OF $\phi$ WITH ADJUSTED DOTCVP RESULTS ........................................... 60
3.6 MODEL PARAMETERS FOR EXAMPLE 3.2 ................................. 63
3.7 RESULTS FOR EXAMPLE 3.2 .................................................. 66
3.8 MODEL PARAMETERS FOR EXAMPLE 3.3 ................................. 68
3.9 RESULTS FOR EXAMPLE 3.3 WITH FIXED ITS TRUNCATION ORDER $\kappa = 17$ AND DIFFERENT TAYLOR MODEL ORDERS $q$ ..................................................... 70
3.10 RESULTS FOR EXAMPLE 3.3 WITH FIXED TAYLOR MODEL ORDER $q = 3$ AND DIFFERENT ITS TRUNCATION ORDERS $\kappa$ ..................................................... 71
3.11 RESULTS FOR EXAMPLE 3.3 WITH $\kappa = 5$ AND $q = 3$ .............. 72
4.1 MODEL PARAMETERS FOR EXAMPLE 4.1 ................................. 86
4.2 MODEL PARAMETERS FOR EXAMPLE 4.2 .................................. 88
4.3 RESULTS FOR EXAMPLE 4.2 .................................................. 89
4.4 MODEL PARAMETERS FOR EXAMPLE 4.3 .................................. 91
4.5 MODEL PARAMETERS FOR EXAMPLE 4.4 .................................. 93
4.6 RESULTS FOR EXAMPLE 4.4 USING DIFFERENT OBJECTIVE TOLERANCE $\epsilon$ ..................................................... 94
4.7 RESULTS FOR EXAMPLE 4.4 USING DIFFERENT PATH CONSTRAINT $S_{low}$ ...................................................... 95
5.1 RESULTS FOR ADLB WITH BALANCED WORKLOAD INITIALIZATION USING 16 PROCESSES ........................................ 126
ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Mark Stadtherr, for the continuous support of my Ph.D study and research, for his patience, motivation and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. I must especially thank him for all the amazing opportunities in conference and in travel that he helped to make possible. I could not have imagined having a better advisor for my Ph.D study.

I thank my current and previous research group colleagues. I thank D. Andrei Măceș for the teamwork in course projects, for the stimulating discussions and for helping me with Linux operating system and programming problems. I must especially thank Youdong Lin for his programming expertise; without his novel research work, I would not have been able to pursue this research in the first place. Thanks also goes to Toni Gau and Gang Xu for inspiring me to explore parallel computation.

Last but not the least, I would like to thank my family: my parents Lu Tao and Zheng Zhao, for giving birth to me at the first place and supporting me spiritually throughout my life.
CHAPTER 1

INTRODUCTION

1.1 Background

Dynamic systems arise naturally in many fields of science and engineering. These systems are often described by ordinary differential equations (ODEs), involving a set of state variables, their rates of change and a set of parameters. In real world industrial applications, it is common that disturbances exist in a dynamic system and/or that an operating range has to be considered, and therefore some parameters and/or initial conditions of an ODE system may not be known with certainty. Simulations of nonlinear dynamic systems with uncertainties are commonly based on the use of classical numerical integration schemes combined with Monte Carlo or other sampling methods. However, such a combination can only offer an approximation to the true range of solutions to the uncertain ODE system. Recently, verified interval solution procedures \[23, 63, 84-86\] were developed to provide rigorous enclosures, which contain all the solutions for every possible value of the uncertain parameters and initial conditions. Interval methods have been successfully applied to a variety of initial value problems \[63\] and boundary value problems \[66\]. More importantly, these methods can serve as a powerful kernel (bounding tool) for global dynamic optimization algorithms and enable one to obtain a guaranteed deterministic global solution. To date, various
applications have been focused on unconstrained optimization with only decision parameters [33, 61, 62]. In this dissertation, we will present new algorithms for two types of problems: dynamic global optimization subject to inequality path constraints, and robust design and global optimization with uncertain parameters.

1.2 Dynamic Global Optimization (DGO) Subject to Inequality Path Constraints (IPC)

In industrial practice, it is often required that dynamic processes be operated to satisfy certain safety and/or quality control concerns over the entire operating time. For example, in the field of chemical engineering, batch and semi-batch processes are often required to be run within certain temperature and/or concentration ranges over their entire reaction time. Optimization problems in such systems are thus subject to appropriate path constraints, which state that the operating trajectory may never enter regions of the state space that could cause process safety or product quality problems. Classical local optimization techniques, combined with traditional numerical integration schemes may be insufficient to solve these path-constrained optimization problems rigorously. Classical local optimization techniques provide no guarantee of finding the global optimum when problems exhibit multiple local optima. Moreover, when integrating with traditional numerical schemes, path constraints can only be examined at the end of an integration step, not during the entire time interval of the step. To address the issue of multiple local optima, various deterministic global optimization approaches have been developed. However, these approaches all focus on the case of unconstrained problem. In contrast, relatively little work has been done on rigorous global optimization for problems with path constraints.
In this dissertation, we present a new strategy for the rigorous global optimization of dynamic systems with inequality path constraints, based on a control parameterization approach. This method is based on Taylor-model constraint propagation under a branch-and-reduce framework [61, 62, 66]. A key feature of the method is that it fully exploits rigorous bounding ability of a verified Taylor model solver for ODEs. The solver [63] consists of two phases applied at each integration step. In phase one, existence and uniqueness of the solution are proven, and a coarse enclosure of the solution for the entire integration step is computed. In phase two, a tighter enclosure of the solution is computed and bounded by Taylor models, which are symbolic (algebraic) functions of the decision variables. In each integration step, our strategy begins with extracting information from the phase one solution to investigate the path constraints over the entire time interval of an integration step, thus providing rigorous assurance that the path constraints are satisfied over the entire time horizon [65]. After phase two, our strategy applies constraint propagation techniques on the explicit algebraic representation to greatly reduce the search space of the decision variables. Furthermore, we put our strategy into practice with the consideration of the following issues:

1. Branch and reduce alone will no longer be sufficient to refine and determine searching region. The effect of integration time step as well as other solver parameter settings must be considered. A heuristic procedure is given to determine appropriate settings for the verified ODE solver (VSPODE).

2. Since the applications may be process safety related, we adopt a conservative inner-subpaving approach to only update solutions on confirmed feasible regions. An outer-subpaving approximation is also maintained for further investigation, if a less conservative approach is justified.
3. An approach can be very inefficient if proceeds by first identifying the domain that satisfies the IPCs and then searching that domain for the global optimum. We intertwine the constraint propagation on objective function test and path constraint test to reduce redundant work.

1.3 Robust Design and Global Optimization for Dynamic Systems

In real world applications, dynamic processes and control systems are often designed in a way that accounts for uncertainties in process parameters and possible disturbances. For example, in chemical engineering, batch and semi-batch processes may often be designed to guarantee a desired yield percentage over a given operating temperature or reactant concentration range. This is a type of robust design problem in which we look for feasible operating regions that can satisfy all the constraints. It is also common that we want to guarantee the best possible yield in the worst-case scenario in the presence of uncertain operating temperatures or reactant concentrations. This then gives a type of robust optimization problem. In this dissertation, we will present a general rigorous solution framework that is applicable to both robust design and optimization for dynamic systems.

In most previous research, the robust design problems are either restricted to linear models [30, 94, 108] or solved with a non-verified approach [50] which may not represent the true dynamics of a nonlinear system. Recently, a rigorous approach was developed for safety analysis [65]. However, this approach treats uncertain parameters as additional design variables, which requires the scan of entire parameter space and thus introduces unnecessary computational cost. In field of robust optimization, algorithmic procedures for deterministic global solution have
been proposed for static problems \cite{125}, based on the use of a general bi-level framework which treats uncertain parameters and design variables differently.

In this dissertation, our solution procedure extends the dynamic global optimization algorithm introduced above. Therefore, it is also based on Taylor-model constraint propagation using a branch-and-reduce framework, and offers continuous satisfaction of inequality path constraints. In order to avoid redundant work, we adopt a bi-level structure for branch-and-reduce. In the inner layer, a design subspace is tested over different subregions of uncertainty. In the outer layer, the results from the inner layer are gathered and analyzed to determine whether to keep or discard a design subspace. Eventually our approach will provide a set of rigorously guaranteed feasible operational (decision) subregions for robust design problems and will locate an $\epsilon$-global solution over the (feasible) inner sub-paving region for robust optimization problems.

1.4 Parallel Computation

In practice, dynamic optimization problems with path constraints and/or robust design requirements can be computationally challenging, especially when solved with a global rigorous approach. Because the global optimization problems considered are NP-hard, requiring a worst-case computation time that may scale exponentially with the problem size, the algorithms discussed here may require hours (on a modern single CPU core) to find a guaranteed solution. This may be tolerable for a lot of design tasks. However a variety of dynamic processes require fast response or instant decision and thus a rigorous global approach may not be potentially applicable for these processes. Therefore, there is the need to speed up the computation for global rigorous method by any possible means. In
this dissertation, another major goal is to discuss how to tackle computational challenge with parallel programming techniques.

Since the end of frequency scaling around 2004, use of parallel computing has become the dominant mechanism for processor performance gains. Nowadays, parallel hardware and architectures, such as multi-core processors and networked cluster servers are readily available. Parallel design patterns and programming tools are also widely available. However, advances in parallel algorithms for rigorous dynamic optimization have not kept pace with advances in hardware and software. Here, for exploiting parallel hardware, we will design a general dynamic load balancing framework, implement two versions of it using Message Passing Interface (MPI) and Posix multi-threads, and test it using a dynamic optimization problem on a multi-core server.

Multiple opportunities exist in implementing parallel computation into global dynamic optimization algorithms. The coarse-grain dynamic load balancing (DLB) approach is chosen here based on three considerations: 1. It is possible to achieve superlinear speedup [42, 124], a speedup of more than \( p \) when using \( p \) computing units (processes or threads), since one unit may find a good solution earlier than in the sequential run and then broadcast it to other units to accelerate their backtracking processes. 2. It is easier to scale up with the increase of problem size and number of computing nodes. Other finer grain options, such as parallelizing polynomial operations or series expansions will be constrained by specific parameter orders. 3. Compared to a fine-grain approach, the DLB approach costs less time in communication between nodes. Although DLB has these potential advantages, it requires careful design to reduce idle waiting states and avoid deadlock. In Chapter 5, we will discuss several DLB design factors, including communication
schemes and the effect of the virtual network, and then compare performance between different options, such as synchronous vs. asynchronous schemes and 1D vs. 2D networks.

1.5 Outline

The rest of this dissertation is structured as follows. In the next chapter, we provide a brief review of interval analysis, Taylor models, constraint propagation, and the verified state bounding method. In Chapter 3, a new approach is described for the rigorous global optimization of dynamic systems with inequality path constraints. We then extend this algorithm to handle robust design and optimization problems with uncertain variables in Chapter 4. In Chapter 5, we present several implementations of a dynamic load balance approach to parallelize the branch and reduce framework. Computational results and discussion are given at the end of these three chapters (3 to 5). Finally, Chapter 6 draws conclusions and gives recommendations for future research.
CHAPTER 2

BACKGROUND

2.1 Interval Analysis

A real interval \( X = [\underline{X}, \bar{X}] \) is defined by \( X = \{ x \in \mathbb{R} \mid \underline{X} \leq x \leq \bar{X} \} \). Here an underline is used to indicate the lower bound of an interval and an overline is used to indicate the upper bound. The width of an interval \( X \) is defined by \( w(X) = \bar{X} - \underline{X} \), while the midpoint is denoted by \( m(X) = (\bar{X} + \underline{X})/2 \). A real interval vector \( \mathbf{X} = (X_i) = (X_1, X_2, \cdots, X_n)^T \) has \( n \) real interval components and can be interpreted geometrically as an \( n \)-dimensional rectangle. The width of an interval vector \( \mathbf{X} \) is the width of its widest element; that is, \( w(\mathbf{X}) = \max_i w(X_i) \). Basic arithmetic operations with intervals are defined by

\[
X \text{ op } Y = \{ x \text{ op } y \mid x \in X, y \in Y \}, \text{ op } \in \{+, -, \times, \div \}.
\]  

(2.1)
The four elementary operations are commonly implemented at the endpoints of $X$ and $Y$ by,

\[
X + Y = [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}] \quad (2.2)
\]

\[
X - Y = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}] \quad (2.3)
\]

\[
X \times Y = [\min(\underline{X} \underline{Y}, \overline{X} \overline{Y}, \underline{X} \overline{Y}, \overline{X} \underline{Y}), \max(\underline{X} \underline{Y}, \overline{X} \overline{Y}, \underline{X} \overline{Y}, \overline{X} \underline{Y})] \quad (2.4)
\]

\[
X \div Y = [\underline{X}, \overline{X}] \times [\frac{1}{\underline{Y}}, \frac{1}{\overline{Y}}] \text{ where } 0 \notin Y. \quad (2.5)
\]

As an example, for $X = [-1, 2]$ and $Y = [-2, -1],

\[
X + Y = [-1 + (-2), 2 + (-1)] = [-3, 1] \quad (2.6)
\]

\[
X - Y = [-1 - (-1), 2 - (-2)] = [0, 4] \quad (2.7)
\]

\[
X \times Y = [\min(2, 1, -4, -2), \max(2, 1, -4, -2)] = [-4, 2] \quad (2.8)
\]

\[
X \div Y = [-1, 2] \times [-1, -\frac{1}{2}] = [-2, 1]. \quad (2.9)
\]

Division in the case of $0 \in Y$ is allowed only in extensions of interval arithmetic [44]. In most implementations of interval arithmetic, division by zero will cause an error or lead to warning and exception throwing, so that users can either terminate the computation and reset input interval range or specify subsequent operations to handle it.

Interval versions of the elementary functions are also defined, based on the monotonicity properties already known. For one variable monotonic functions, the value range can be bounded simply by the function value at two endpoints. For example,
\[ a^X = [a^X, a^X], \quad a > 1 \quad (2.10) \]
\[ a^X = [a^X, a^X], \quad 0 < a < 1 \]
\[ \log_a(X) = [\log_a(X), \log_a(X)], \quad a > 1, \quad X > 0 \quad (2.11) \]
\[ \log_a(X) = [\log_a(X), \log_a(X)], \quad 0 < a < 1, \quad X > 0 \]
\[ X^n = [X^n, X^n], \quad \text{for odd } n \in \mathbb{N} \quad (2.12) \]

For piecewise monotonic functions (e.g. sin, cos, etc.), the value range can be obtained by considering pre-calculated function values at critical points (local minimum or maximum) as well as the two boundaries. For instance, even power operation is defined by,

\[
X^n = \begin{cases} 
[X^n, X^n], & X \geq 0 \\
[X^n, X^n], & X \leq 0 \quad \text{for even } n \in \mathbb{N} \\
[0, \max(X^n, X^n)], & \text{otherwise} 
\end{cases} \quad (2.13)
\]

In real-life machine operations, interval arithmetic also accounts for the error in binary floating-point computation, by applying directed (outward) rounding on endpoints; that is, the lower bound is rounded down and the upper bound is rounded up. In this way, interval operations are guaranteed to produce bounds that are rigorous both mathematically and computationally. A number of good introductions to interval analysis and computing with intervals are available [44, 54, 82, 89].

For a real function \( f(\mathbf{x}) \), the interval extension \( F(\mathbf{X}) \) encloses the range of \( f(\mathbf{x}) \) for \( \mathbf{x} \in \mathbf{X} \). A simple way to compute bounds on the range of \( f(\mathbf{x}) \) is to
substitute the given interval $X$ into the expression for $f(x)$ and then to evaluate with interval arithmetic and elementary functions. However, the tightness of the bounds in this “natural” interval extension depends on the form of the expression used to evaluate $f(x)$. If this is a single-use expression, in which no variable appears more than once, then the exact function range will be obtained (within roundout). However, if any variable appears more than once, then overestimation of the range may occur. Such overestimation is due to the “dependency” problem of interval arithmetic. While a variable may take on any value within its interval, it must take on the same value each time it occurs in an expression. However, this type of dependency is not detected when the natural interval extension is computed. For example,

\[ f(x) = \frac{x}{x^2 + 1}, \quad x \in [0, 1] \quad (2.14) \]

\[ f(x) \in [0, \frac{1}{2}] \]

Apply direct substitution:

\[ F(X) = \frac{[0, 1]}{[0, 1] \times [0, 1] + 1} \]
\[ = [0, 1]/[1, 2] \]
\[ = [0, 1] \]

The rule of thumb to address dependency is the use of subdividing method which iteratively bisects an initial interval variable space and then evaluating the interval bound of function on each subregion. The union of these interval bounds is often tighter than the enclosure obtained from computing with the entire original interval. For example, still consider eq [2.14] if we evenly divide $X$ into 4 pieces \{[0, \frac{1}{4}], [\frac{1}{4}, \frac{1}{2}], [\frac{1}{2}, \frac{3}{4}], [\frac{3}{4}, 1]\} and bound $f(x)$ over each subinterval,
we obtain \(\{[0, \frac{1}{3}], \left[\frac{1}{5}, \frac{8}{17}\right], \left[\frac{8}{25}, \frac{3}{5}\right], \left[\frac{3}{8}, \frac{16}{25}\right]\}\) and therefore \(F(x) \subset [0, \frac{16}{25}]\). This bound is tighter than \([0, 1]\). Subdividing is widely applied together with interval related methods in rigorous analysis and deterministic optimization problems.

In certain cases, the dependent subtraction operation (also called as cancellation operation) can be used to avoid dependency. Assume that an interval \(S\) depends additively on another interval \(A\). The dependent subtraction operation is defined by \(S \ominus A = [S - A, S - \overline{A}]\). For example, let \(A = [0, 3], B = [1, 3], C = [3, 5]\) and \(S = A + B + C = [4, 11]\). Consider that only \(S\) is stored in memory and that later one wants to evaluate \(A + B\) by subtracting \(C\) from \(S\). Using the standard subtraction operation yields overestimated range \(S - C = [4, 11] - [3, 5] = [-1, 8]\) for \(A + B\). Using the dependent subtraction operation yields the exact range \(S \ominus C = [4, 11] \ominus [3, 5] = [1, 6]\) for \(A + B\).

Another source of overestimation that may occur in the use of interval methods is the “wrapping” effect. This occurs when an interval vector (rectangle) is used to enclose (wrap) a set of results that is not an interval. For example, consider the region \(\mathcal{R} = \{x^2 + y^2 \leq 1, x \geq 0, y \geq 0\}\) demonstrated in Fig \ref{fig:2.1} by the shade area. The tightest bounding interval for \(\mathcal{R}\) is \(\{X, Y\} = \{[0, 1], [0, 1]\}^T\). However, this interval vector includes extra subspace which is not part of \(\mathcal{R}\), as shown by the solid area in Figure \ref{fig:2.1}. Assume that the function \(f(x, y) = x + y\) will be evaluated. If the computation is taken over \(\{X, Y\} = \{[0, 1], [0, 1]\}\)^T instead of \(\mathcal{R}\), this leads to an overestimated range \(F(X, Y) = [0, 2]\), whereas the actual range equals to \([0, \sqrt{2}]\). One approach that can be used to address both the dependency problem and the wrapping effect is the use of Taylor models.
2.2 Taylor Models

Makino and Berz [72, 73] have described a remainder differential algebra (RDA) approach for bounding function ranges and controlling the dependency problem of interval arithmetic. In this method, a function is represented by a model consisting of a Taylor polynomial expression and an interval remainder bound.

One way of forming a Taylor model of a function is by using the Taylor theorem. Consider a real function \( f(x) \) that is \((q+1)\) times partially differentiable for \( x \in X \) and let \( x_0 \in X \). The Taylor theorem states that for each \( x \in X \), there exists a real \( \zeta \) with \( 0 < \zeta < 1 \) such that

\[
f(x) = \sum_{i=0}^{q} \frac{1}{i!} [(x - x_0) \cdot \nabla]^i f(x_0) + \frac{1}{(q+1)!} [(x - x_0) \cdot \nabla]^{q+1} f [x_0 + (x - x_0)\zeta],
\]

Figure 2.1. Example of the wrapping effect
where the partial differential operator \([g \cdot \nabla]^k\) is

\[
[g \cdot \nabla]^k = \sum_{j_1 + \cdots + j_m = k}^{k!} \frac{k!}{j_1! \cdots j_m!} g_1^{j_1} \cdots g_m^{j_m} \frac{\partial^k}{\partial x_1^{j_1} \cdots \partial x_m^{j_m}}.
\] (2.16)

where \(p_f\) is a \(q\)-th order polynomial (truncated Taylor series) in \((x - x_0)\) and \(r_f\) is a remainder, which can be quantitatively bounded over \(0 < \zeta < 1\) and \(x \in X\) using interval arithmetic or other methods to obtain an interval remainder bound \(R_f\). A \(q\)-th order Taylor model \(T_f(x) = p_f(x - x_0) + R_f\) for \(f(x)\) over \(X\) then consists of the polynomial expression \(p_f\) and the interval remainder bound \(R_f\) and is denoted by \(T_f = (p_f, R_f)\). The expression \(f(x)\) can now be bounded for \(x \in X\) by seeking bounds on the Taylor model \(T_f(x)\) for \(x \in X\). This could be done using the natural interval extension \(T_f(X)\), but usually tighter bounds can be obtained using other methods [63, 75, 76, 90]. Following Lin and Stadtherr [63], we use an approach on \(p_f\) in which the sum of all linear and diagonal quadratic terms is bounded exactly, with all other terms in \(p_f\) bounded directly using interval arithmetic. That is,

\[
B(p) = \sum_{i=1}^{m} \left[ a_i (X_i - x_{i0})^2 + b_i (X_i - x_{i0}) \right] + Q,
\] (2.17)

where \(Q\) is the interval bound of all remaining terms, and is evaluated with interval arithmetic. In eq [2.17], \(X_i\) occurs in both linear and quadratic terms, and therefore a dependency problem exists. For \(|a_i| \geq \omega\), where \(\omega\) is a user defined small positive number, eq [2.17] can be rearranged such that each \(X_i\) occurs only once; that is,

\[
B(p) = \sum_{i=1}^{m} \left[ a_i \left( X_i - x_{i0} + \frac{b_i}{2a_i} \right)^2 - \frac{b_i^2}{4a_i} \right] + Q.
\] (2.18)
In this way, the dependence problem in bounding $p_f$ is possibly alleviated so that a sharper bound may be obtained. If $|a_i| < \omega$, the reformulation step can not have significant impact on the bounding performance and thus direct interval evaluation will be used instead.

In practice, it is more useful to compute Taylor models of functions by performing Taylor model operations. Arithmetic operations with Taylor models can be done using RDA operations [72-74], which include addition, multiplication, reciprocal and intrinsic functions. Let $T_f$ and $T_g$ be the Taylor models of the functions $f(x)$ and $g(x)$ respectively over the interval $x \in X$. The Taylor model of $f \pm g$ can be represented as

$$T_{f \pm g} = (p_f, R_f) \pm (p_g, R_g) = (p_f \pm p_g, R_f \pm R_g),$$

where $p_f \pm p_g$ is the sum of two $q$-th order polynomials, and $R_f \pm R_g$ is the new remainder bound. The Taylor model for the product $f \times g$ is

$$T_{f \times g} = (p_f, R_f) \times (p_g, R_g) = p_f \times p_g + p_f \times R_g + p_g \times R_f + R_f \times R_g.$$  

Note that $p_f \times p_g$ is a polynomial of order $2q$. In order to be consistent with the $q$-th order polynomial in the Taylor model operands, this term is split into a polynomial $p_{f \times g}$ of up to $q$-th order, and an extra polynomial $p_e$. Thus, the remainder bound of $f \times g$ can be found using

$$R_{f \times g} = B(p_e) + B(p_f) \times R_g + B(p_g) \times R_f + R_f \times R_g,$$

where $B(p) = P(X - x_0)$ is an interval bound of the polynomial $p(x - x_0)$ over $x \in X$. Similarly, an interval bound on an overall Taylor model $T = (p, R)$ will
be denoted by $B(T) = B(p) + R$. Using these operations, it is possible to begin
with simple functions such as the constant $f(x) = k$, for which $T_f = (k, [0, 0])$, and
the identity $f(x_i) = x_i$, for which $T_f = (x_{i0} + (x_i - x_{i0}), [0, 0])$, and to then
compute Taylor models for more complicated functions.

For example, a third order Taylor model for $f(x) = x \ln(x) + x$ over interval
$x \in [0.1, 0.2]$ is computed as follows. Variable $x$ is initialized by identity Taylor
model $T_x = x_0 + (x - x_0) + [0, 0]$ with $x_0 = m(X) = 0.15$. Let $x_p = x - x_0$, $x_p \in [-0.05, 0.05]$. The Taylor model of $\ln(x)$ is obtained from Taylor theorem,

$$\ln(x) = \ln(x_0) + x_p \frac{1}{x_0} + \frac{1}{2} x_p^2 \frac{-1}{x_0^2} + \frac{1}{6} x_p^3 \frac{-2}{x_0^3} + \frac{1}{24} x_p^4 \left(\frac{-6}{\zeta^4}\right)$$

for $\zeta \in [0.1, 0.2]$. The interval remainder term is then evaluated by direct sub-
stitution and natural interval arithmetic. We compute the polynomial coefficient
and obtain a third order Taylor model,

$$T_{\ln(x)} = -1.8971 + 6.6667 \ x_p - 22.2222 \ x_p^2 + 98.7654 \ x_p^3 + [-0.0156, 0.0]. \ (2.23)$$

$T_{x\ln(x)}$ is computed from multiplication of two Taylor models $T_{\ln(x)} \times T_x$,

$$T_{x\ln(x)} = (-1.8971 + 6.6667 \ x_p - 22.2222 \ x_p^2 + 98.7654 \ x_p^3 + [-0.0156, 0.0]) \times (0.15 + x_p + [0, 0]) \ (2.24)$$

$$= (-0.2845 + x_p - 3.3333 \ x_p^2 + 14.8148 \ x_p^3) +$$

$$(-1.8971 \ x_p + 6.6667 \ x_p^2 - 22.2222 \ x_p^3) +$$

$$([-0.0156, 0.0] \times (0.15 + x_p) + 98.7654 \ x_p^4)$$

$$= -0.2845 - 0.8971 \ x_p + 3.3333 \ x_p^2 - 7.4074 \ x_p^3 + [-0.0025, 0.0006]$$
Eventually, we add \( T_x \) to \( T_x \ln(x) \).

\[
T_{x \ln(x)+x} = (-0.2845 - 0.8971 \ x_p + 3.3333 \ x_p^2 - 7.4074 \ x_p^3
\]
\[
+[-0.0025, 0.0006]) + (0.15 + x_p + [0, 0])
\]
\[
= -0.1345 + 0.1029 \ x_p + 3.3333 \ x_p^2 - 7.4074 \ x_p^3 + [-0.0025, 0.0006]
\]

The result is the Taylor model of \( T_{x \ln(x)+x} \) over \( x \in [0.1, 0.2] \). Using the procedure given above in eq 2.17, we obtain the interval bound of \( B(T_{x \ln(x)+x}) = [-0.1388, -0.1195] \). This solution enclosure is sharper than the one obtained from standard interval arithmetic

\[
[0.1, 0.2] \ln([0.1, 0.2]) + [0.1, 0.2] = [0.2 \ln(0.1) + 0.1, 0.1 \ln(0.2) + 0.2]
\]
\[
= [-0.3605, 0.0391]. \quad (2.26)
\]

The actual bound is \([f(\frac{1}{e^2}), f(0.2)] = [-0.1354, -0.1218]\), which is enclosed by \( B(T_{x \ln(x)+x}) \).

It has been reported that, compared to other rigorous range-bounding techniques, the use of Taylor models often provides tighter enclosures for functions with modest to complicated dependencies \([72, 73, 90]\). The applications and limitations of Taylor models are discussed in more detail elsewhere \([90]\).

2.3 Constraint Propagation with Taylor Models

The scheme \([63]\) used in the bounding of Taylor models can be exploited in performing constraint propagation with Taylor models. Information provided by a constraint can be used to eliminate incompatible values from the domain of its variables. This domain reduction can then be propagated to all constraints.
on that variable, where it may be used to further reduce the domains of other variables. This is the process known as constraint propagation [44, 54, 56]. For example, consider a separable quadratic constraint [31]

\[-x_1^2 + 4x_1 - x_2 \geq -5\]  
\[x_1 \in [-6, 4]\]  
\[x_2 \in [0, \infty].\]

The constraint propagation is performed in the following steps:

1. Given that \(x_1 \in [-6, 4]\) and \(x_2 \in [0, \infty]\), we know that \(-x_1^2 + 4x_1 \in [-48, 4]\) and \(-x_2 \in [-\infty, 0]\).

2. Therefore \(-x_1^2 + 4x_1 - x_2 \leq 4\) holds.

3. We find that \(-x_1^2 + 4x_1 \geq -5\) from \(x_2 \in [0, \infty]\).

4. Since \(4 \geq -x_1^2 + 4x_1\), the inequality \(-x_2 \geq -5 - 4\) holds.

5. From \(-x_1^2 + 4x_1 \geq -5\), we also have \(x_1 \in [-1, 5]\).

6. Finally, we obtain \(-x_1^2 + 4x_1 - x_2 \in [-5, 4]\) with \(x_1 \in [-1, 4]\) and \(x_2 \in [0, 9]\).

Constraint propagation procedure (CPP) is widely used in various forms in connection with interval methods. It has been shown previously how efficient constraint propagation schemes, based on hull consistency and using Taylor models, can be developed for inequality constraints [61], bound constraints [64], and equality constraints [66]. In this dissertation, we use a propagation procedure based on solving separable quadratic constraint with Taylor models [61].
For each inequality constraint \( c(\mathbf{x}) \leq 0 \), let \( T_c \) be the Taylor model of the constraint function \( c(\mathbf{x}) \) over the interval \( \mathbf{x} \in \mathbf{X} \). The constraint propagation procedure (CPP) starts with an interval bounds test. There are three possible outcomes:

1. If \( B(T_c) > 0 \), then no \( \mathbf{x} \in \mathbf{X} \) can satisfy the constraint; thus, the CPP terminates and the entire region \( \mathbf{X} \) will be discarded.

2. If \( B(T_c) \leq 0 \), then every \( \mathbf{x} \in \mathbf{X} \) will satisfy the constraint; thus \( \mathbf{X} \) cannot be reduced and the CPP can be stopped.

3. Otherwise, at least part of the interval \( \mathbf{X} \) may be eliminated; thus the CPP continues, using an approach based on the range bounding strategy for Taylor models described above.

For some \( i \in \{1, 2, \ldots, m\} \), the polynomial coefficients of the terms \( (x_i - x_{i0})^2 \) and \( (x_i - x_{i0}) \) of \( T_c \) are denoted by \( a_i \) and \( b_i \) respectively. Here, \( x_{i0} \) is the midpoint \( m(X_i) \) and the value is fixed during the CPP. According to eq \( 2.18 \), if \( |a_i| \geq \omega \), the bounds on \( T_c \) can be expressed as

\[
B(T_c) = B(p) + R = a_i \left( X_i - x_{i0} + \frac{b_i}{2a_i} \right)^2 - \frac{b_i^2}{4a_i} + S_i,
\]

where \( S_i \) is obtained by dependent subtraction using

\[
S_i = B(T_c) \ominus \left[ a_i \left( X_i - x_{i0} + \frac{b_i}{2a_i} \right)^2 - \frac{b_i^2}{4a_i} \right].
\]

Now define the intervals \( U_i = X_i - x_{i0} + \frac{b_i}{2a_i} \) and \( V_i = \frac{b_i^2}{4a_i} - S_i \), so that \( B(T_c) = a_i U_i^2 - V_i \). The goal is to identify and retain only the part of \( X_i \) that contains values of \( x_i \) for which it is possible to satisfy \( c(\mathbf{x}) \leq 0 \). Since \( B(T_c) \) bounds the
range of $c(x)$ for $x \in X$, the constraint $c(x) \leq 0$ will be satisfied if $B(T_c) \leq 0$. Thus, to identify bounds on the part of $X_i$ that satisfies the constraint, we can use the condition

$$a_i U_i^2 \leq V_i. \quad (2.30)$$

Then, the set $U_i$ that satisfies eq (2.30) can be determined to be

$$U_i = \begin{cases} \emptyset & \text{if } a_i > 0 \text{ and } V_i < 0 \\ \left[-\sqrt{\frac{V_i}{a_i}}, \sqrt{\frac{V_i}{a_i}}\right] & \text{if } a_i > 0 \text{ and } V_i \geq 0 \\ [-\infty, \infty] & \text{if } a_i < 0 \text{ and } V_i \geq 0 \\ \left[-\infty, -\sqrt{\frac{V_i}{a_i}}\right] \cup \left[\sqrt{\frac{V_i}{a_i}}, \infty\right] & \text{if } a_i < 0 \text{ and } V_i < 0 \end{cases}. \quad (2.31)$$

The part of $X_i$ to be retained is then $X_i = X_i \cap \left(U_i + x_{i0} - \frac{b_i}{2a_i}\right)$.

If $|a_i| < \omega$, then eq (2.18) cannot be used, but eq (2.17) can. Following a procedure similar to that used above, we now define $U_i = X_i - x_{i0}$ and $V_i = -a_i (X_i - x_{i0})^2 - S_i$. To identify bounds on the part of $X_i$ that satisfies the constraint, we can now use the condition

$$b_i U_i \leq V_i. \quad (2.32)$$

Then, the set $U_i$ that satisfies eq (2.32) can be determined to be

$$U_i = \begin{cases} \left[-\infty, \frac{V_i}{b_i}\right] & \text{if } b_i > 0 \\ \left[\frac{V_i}{b_i}, \infty\right] & \text{if } b_i < 0, \end{cases} \quad (2.33)$$

where it is assumed that $|b_i| \geq \omega$. The part of $X_i$ to be retained is then $X_i = X_i \cap (U_i + x_{i0})$. If both $|a_i|$ and $|b_i|$ are less than $\omega$, then no CPP will be applied...
on $X_i$.

Using this constraint propagation procedure (CPP) with a Taylor model $T_c$ of the constraint, a region $X_{nf} \subseteq X$ that is \textit{guaranteed} not to satisfy the constraint may be identified and removed from $X$ to obtain an updated $X$; that is, $X \leftarrow X \setminus X_{nf} = \{x \in X \mid x \notin X_{nf}\}$. The region $X_{nf}$ is not necessarily an interval, but a set of intervals generated in such a way that $X \setminus X_{nf}$ is an interval or an empty set. If the reduction in the size of $X$ is sufficient (more than 10% reduction in volume is the heuristic used in our implementation), then the CPP is repeated. In repeating the CPP, we could use VSPODE to recompute the Taylor model of the constraint over the smaller $X$; however, in our experience, this additional expense is usually not justified, as useful domain reduction may be obtained by continuing to use the initial Taylor model. The final outcome of the CPP may be that $X$ is completely eliminated (becomes an empty set), or that $X$ is reduced in size, or that $X$ is not changed at all.

2.4 Bounding State Variables

When a standard sequential approach is applied to the dynamic optimization problem, the objective function $\phi [x(t_\eta, \theta), \theta; \eta = 0, 1, \ldots, r]$ is evaluated, for a given value of $\theta$, by applying an ODE solver to the constraints to eliminate $x(t_\eta, \theta), \eta = 1, \ldots, r$. In the deterministic global optimization algorithm described here, we will use a branch-and-bound approach that requires the evaluation of bounds on $\phi$, given some bounds $\theta \in \Theta$ on the adjustable parameters. Thus, as emphasized above, a key step is the determination of bounds on the states $x(t_\eta, \theta), \eta = 1, \ldots, r$ for $\theta \in \Theta$. For this purpose, we will use the method described by Lin and Stadtherr \cite{63}, and implemented in a code called VSPODE. This is an
interval method \([84, 88]\) (such methods are also called validated or verified methods) and also uses Taylor models. It is capable of determining mathematically and computationally guaranteed bounds on the state variables for nonlinear ODE systems with interval-valued parameters and/or initial states. We will briefly described the method here. Another interesting approach using Taylor models for state bounding has been described recently by Sahlodin and Chachuat \([100]\).

Focusing on the ODE constraint \(\dot{x} = f(x, \theta), x_0 = x_0(\theta)\), the goal is to determine bounds on the state variables \(x\) for \(\theta \in \Theta\) and \(x_0 \in X_0(\Theta)\). We denote by \(x(t; t_j, X_j, \Theta)\) the set of solutions \(x(t; t_j, x_j, \theta) = \{x(t; t_j, x_j, \theta) | x_j \in X_j, \theta \in \Theta\}\), where \(x(t; t_j, x_j, \theta)\) denotes a solution of \(\dot{x} = f(x, \theta)\) for the initial condition \(x = x_j\) at \(t_j\). As one outcome of the method outlined here, we will determine enclosures \(X_j\) of the state variables at a series of time steps \(j = 1, \ldots, r\), such that \(x(t_j; t_0, X_0, \Theta) \subseteq X_j\).

2.4.1 Phase One

Assume that at \(t_j\) we have an enclosure \(X_j\) of \(x(t_j; t_0, X_0, \Theta)\), and that we want to carry out an integration step with stepsize \(h_j = t_{j+1} - t_j\) to compute the next enclosure \(X_{j+1}\). Then, in the first phase (verification phase) of the method, the goal is to find an \textit{a priori} enclosure \(\tilde{X}_j\) of the solution such that a unique solution \(x(t; t_j, x_j, \theta) \in \tilde{X}_j\) is guaranteed to exist for all \(t \in [t_j, t_{j+1}]\), all \(x_j \in X_j\), and all \(\theta \in \Theta\). This is done with a high-order interval Taylor series (ITS) with respect to time, using the Picard-Lindelöf operator and Banach fixed-point theorem. That is, we determine \(\tilde{X}_j\) such that for \(X_j \subseteq \tilde{X}^0_j\),

\[
\tilde{X}_j = \sum_{i=0}^{\kappa-1} [0, h_j]^i F^{|i|}(X_j, \Theta) + [0, h_j]^\kappa F^{|\kappa|}(\tilde{X}^0_j, \Theta) \subseteq \tilde{X}^0_j.
\]  

(2.34)
Here $\kappa$ denotes the order of the truncation error, $\tilde{X}_j^0$ is an initial estimate of $\tilde{X}_j$, and the coefficients $F^{[i]}$ are interval extensions of the Taylor coefficients $f^{[i]}$ of $x(t)$ with respect to time, which can be obtained recursively in terms of $\dot{x}(t) = f(x, \theta)$ using automatic differentiation. This is an extension to parametric ODEs of the approach used in the popular VNODE method \cite{84, 87} for verified solution of ODEs. Satisfaction of eq. (2.34) verifies that there exists a unique solution $x(t; t_j, x_j, \theta) \in \tilde{X}_j$ for all $t \in [t_j, t_{j+1}]$, all $x_j \in X_j$, and all $\theta \in \Theta$.

Note that $\tilde{X}_j$ rigorously encloses all the possible values of the state variables during an entire integration step, not only at $t_{j+1}$. We will use this enclosure in the optimization algorithm described below to enforce continuous satisfaction of the IPCs. In the usual VSPODE implementation \cite{63} of this method, there are multiple options for the stepsize. An automatic stepsize adjustment procedure \cite{84} is provided that will attempt to use a relatively large $h_j$. Alternatively, a fixed stepsize $h$ can be specified by the user. However, this stepsize will be accepted as $h_j$ at time $t_j$ only if it results in satisfaction of eq. (2.34). Otherwise, VSPODE can be set either to continue with automatic adjustment, which will shrink the stepsize but still try to find a stepsize as large as possible, or to start over using a stepsize that has been reduced by some user-specified fraction. For the application of interest here, however, we find it useful to treat the stepsize adjustment procedure externally as part of the optimization algorithm; thus VSPODE will not adjust the stepsize and it is possible (if $h_j$, $X_j$ or $\Theta$ is too large) for the verification stage to fail (eq. (2.34) cannot be satisfied). If such a failure occurs, it is dealt with in the optimization algorithm.
2.4.2 Phase Two

In the second phase of the method, a tighter enclosure $X_{j+1} \subseteq \tilde{X}_j$ is computed, such that $x(t_{j+1}; t_0, X_0, \Theta) \subseteq X_{j+1}$. This is done by using an ITS approach to compute a Taylor model $T_{x_{j+1}}$ of $x_{j+1}$ in terms of the parameters $\theta$. To do this, the parameters are first expressed as a Taylor model identity function $T_\theta$. Then, Taylor models $T_{f[i]}$ of the interval Taylor series coefficients $f^{[i]}(x_j, \theta)$ can be determined by using RDA operations to compute $T_{f[i]} = f^{[i]}(T_{x_j}, T_\theta)$. Using an interval Taylor series for $x_{j+1}$ with coefficients given by $T_{f[i]}$, and using the mean value theorem, one can obtain a result for $T_{x_{j+1}}$ in terms of the parameters $\theta$. In this process, the wrapping effect is reduced by using a new type of Taylor model that uses a parallelepiped (rather than interval) remainder bound. The resulting Taylor model $T_{x_{j+1}}(\theta)$ can then be bounded over $\theta \in \Theta$ to obtain $X_{j+1}$. Complete details of the computation of $T_{x_{j+1}}$ are given by Lin and Stadtherr.

2.4.3 Performance

As seen in eq. (2.34) for the phase one enclosure $\tilde{X}_j$, the parameter interval $\Theta$ and the stepsize interval $[0, h_j]$ appear repeatedly in expressions that must be evaluated using interval arithmetic during execution of the VSPODE algorithm. Thus, overestimation of bounds resulting from the dependency problem (Section 2.1) of interval arithmetic is inevitable. In the case of a relatively large parameter interval and/or a coarse stepsize, the propagation from time step to time step of relatively loose state enclosures may ultimately lead to verification failure in the first phase (eq. (2.34) can no longer be satisfied or requires a stepsize $h_j$ that is impractically small). In the global optimization application, the parameter interval represents the decision variable space to be searched, and this may initially
be quite large. However, since we will use a branch-and-bound framework for the global optimization algorithm, resulting in repeated subdivision of the search domain, the parameter intervals that must be considered will quickly reach a manageable size. Even if the parameter interval has zero width, some overestimation of the state bounds will still occur due to the stepsize interval. While, as discussed above, there are automatic stepsize adjustment procedures available in VSPODE, we prefer, for purposes of the constrained optimization problem considered here, to take more direct control of the stepsize. To do this, we will select a fixed stepsize \( h \) based on the ability to achieve a user-specified resolution in the constraint values. This process will be implemented as part of an optional “autotuning” procedure for the optimization algorithm, to be discussed in Section 3.3.5.

Two other settings in VSPODE that will affect performance of the optimization procedure are the ITS truncation order \( \kappa \) and the polynomial order \( q \) of the Taylor models used. In general, use of a relatively high \( \kappa \) will provide a smaller truncation error and thus permit the use of a larger stepsize in satisfying eq. (2.34). However, the computational cost of phase one is roughly proportional to \( \kappa \), so there is also motivation to use a \( \kappa \) that is not unnecessarily large. The use of Taylor models to propagate the state enclosures from one time step to the next allows many of the necessary operations to be performed on the symbolic part of the Taylor model, rather than on intervals, thereby significantly reducing overestimation due to interval dependency and wrapping. Thus, a larger Taylor model order \( q \) (more symbolic terms) may lead to reduced propagation of overestimation from time step to time step. This is important since it may delay, or even prevent, the accumulation of bound overestimates that may lead to verification failure, thus extending the time over which the problem can be successfully
integrated. However, the computational cost of computing with Taylor models increases significantly with increasing $q$ due to the increasing number of terms in the polynomial part of the Taylor model. For example, if there are $p = 4$ parameters and $q = 3$, then the number of polynomial terms is 35, but for $q = 5$ this increases to 126 terms (the number of terms in a polynomial of order $q$ in $p$ variables is $(q + p)!/(q!p!)$). As part of the autotuning procedure, we will use a relatively large $\kappa$ and $q$ while determining the stepsize $h$. Then, once $h$ has been determined, we will try to reduce $\kappa$ and $q$ while maintaining satisfaction of the user-specified constraint resolution.
CHAPTER 3

RIGOROUS GLOBAL OPTIMIZATION FOR DYNAMIC SYSTEMS
SUBJECT TO INEQUALITY PATH CONSTRAINTS

3.1 Introduction

Optimization problems involving dynamic systems arise in many application areas in engineering and science. In practice it is often required that dynamic processes be operated to satisfy certain safety and/or quality control concerns over their entire operating time. Optimization problems in such systems are thus subject to appropriate inequality path constraints, which state that the operating trajectory may never enter regions of the state space that could cause process safety, product quality, or other problems. We will describe here a new approach for the rigorous global optimization of dynamic systems with inequality path constraints.

In optimization problems for dynamic systems, one or more of the decision variables may be a continuous function of time, resulting in essentially infinite-dimensional problems. To address this, there are in general two categories of methods. The classical *indirect* methods are based on solving Pontryagin’s necessary conditions [16, 95]. This often results in a two-point boundary value problem, and this may be as difficult to solve as the original optimization problem. In the other category are *direct* methods, which are based on problem formulations that use
discretization to reduce the infinite-dimensional problem to a finite-dimensional one. There are two main approaches to the discretization, the *sequential* approach, in which only the decision variables (control parameters) are discretized \[14, 37, 43, 105, 115, 116\], and the *simultaneous* approach, in which both decision and state variables are discretized \[9, 10, 91, 111, 113\]. Direct methods enable easy incorporation of powerful numerical integration routines for systems of ordinary differential equations (ODEs) or differential-algebraic equations (DAEs) and of nonlinear programming (NLP) routines for optimization. Thus both the sequential and simultaneous approaches have been successfully applied to a wide range of problems, and both are in active use. Recently, for example, the optimization of periodic adsorption processes has been considered by Vetukuri et al. \[117\] using the sequential approach and by Agarwal et al. \[1, 2\] using the simultaneous approach. Features of the two approaches can also be combined into a quasi-sequential approach \[6, 49\]. In this chapter, only the sequential approach will be considered.

The rigorous solution of dynamic optimization problems with inequality path constraints is challenging for two primary reasons. First, the presence of non-convexities may lead to multiple local optima. This may become an issue for both constrained and unconstrained problems, and arises even in relatively simple problems \[71\]. Second, the inequality path constraints are continuously imposed; that is, they must be satisfied at all points in time, not just at discrete points. However, current methods (sequential or simultaneous) directly enforce the constraints only at the level of the discretization used in the problem formulation or in the numerical routines used. Thus, rigorous, continuous satisfaction of the inequality path constraints remains an issue.
To handle the issue of multiple local optima, one approach is to use stochastic global optimization methods. For example, Banga et al. [4] have addressed constrained dynamic optimization problems using various stochastic global optimization methods, as well as a hybrid method that uses a stochastic approach globally with a gradient-based method locally, and this approach has been implemented in the MATLAB toolbox DOTcvp [48]. However, stochastic methods provide no theoretical guarantee of finding the global optimum. Thus there has also been significant recent interest, focused so far on unconstrained problems, in deterministic global optimization (DGO) methods designed to offer some such guarantee. An excellent review of DGO methods, covering many applications, has recently been provided by Floudas and Gounaris [39]. DGO methods are usually constructed on a branch-and-bound framework; thus, the rigor and efficiency (computational cost and bound tightness) of the bounding step has become a core issue, determining the overall performance of DGO methods. Convex relaxation methods for the bounding step have been described by Chachuat and Latifi [18], Papamichail and Adjiman [92, 93], and Esposito and Floudas [34, 35]. Both Chachuat and Latifi [18], and Papamichail and Adjiman [92, 93] obtain a theoretical guarantee of $\epsilon$-global optimality, though at a very high computational cost. Singer and Barton [105] developed an approach, based on use of convex and concave relaxations to bound the state trajectories, that achieves $\epsilon$-global optimality at significantly lower cost. Other recent ideas [99, 101, 104] for constructing convex and concave bounds of the states also may become important in the context of dynamic optimization. Lin and Stadtherr [61, 62] developed a DGO algorithm based on using interval analysis and Taylor models to determine the state bounds [63]. Using this approach it is possible to obtain a rigorous guarantee of either
exact global optimality or \( \epsilon \)-global optimality \[61\]. The former requires the use of interval-Newton methods applied to the sensitivity equations. Either type of guarantee can be obtained at a computational cost that is often much less than that required by the approach of Singer and Barton \[105\]. The DGO methods discussed above are focused on unconstrained problems. In the work described below, we extend the DGO approach of Lin and Stadtherr to problems involving IPCs.

For addressing the satisfaction of IPCs in dynamic systems, most existing methods use certain types of transformations of the original problem. For example, a penalty function \[15, 123\] may be introduced, either added directly to the objective function, or used to determine a set of end point constraints. Another approach is to introduce squared slack variables \[52, 53\] to transform the inequality constraints into equalities. The interior point approach transforms the continuous IPCs into a series of discrete point constraints \[21, 116\]. Hybrid interior point and penalty function approaches have also been proposed \[19, 67\]. Feehery and Barton \[36, 37\] have developed a type of active set strategy, which detects dynamically whether the IPC is active or inactive, and solves either the appropriate unconstrained or equality constrained problem. With any of these approaches, there is no assurance that the IPC will be satisfied between discretization points, which may occur directly in the problem formulation (e.g., interior point approach) and/or in the numerical integration routines used to solve the underlying ODE or DAE problems. In the method described below, we will deal with this issue and present an approach that rigorously guarantees continuous, not just discrete, satisfaction of the IPCs.

In this chapter, we describe a new method for the rigorous global optimization
of dynamic systems subject to inequality path constraints. This method employs
the sequential approach and is based on the use of techniques [63] developed for
the verified solution of parametric ODE systems. These techniques provide rig-
orous interval bounds on the state variables, and thus on the IPCs and objective
function in the dynamic optimization problem. Furthermore, these techniques
provide explicit analytic representations (Taylor models) of these bounds in terms
of the decision variables in the optimization problem. This facilitates the use of
constraint propagation techniques that can greatly reduce the space to be searched
for the global optimum. Through this approach, the search for the global optimum
is restricted to a space in which continuous satisfaction of the IPCs is rigorously
guaranteed, and an $\epsilon$-global optimum within this space is determined. The poten-
tial of this approach is demonstrated, and its performance studied, through the
use of a number of computational examples.

3.2 Problem Statement

In this section, the mathematical formulation of the optimization problem to
be considered is given. Assume that the system to be optimized is described by
an initial value problem (IVP) with the nonlinear ODE model $\dot{x} = f(x, \theta)$ and
initial state $x(t_0) = x_0(\theta)$. Here $x$ is the vector of state variables (length $n$)
and $\theta$ is a vector (length $p$) of adjustable parameters. The final state is denoted
as $x_f = x(t_f)$. The parameter vector $\theta$ includes parameters that appear in the
equations for $\dot{x}$ and that appear in the statement of initial conditions (e.g., one
or more of the initial states may be an adjustable parameter). For the example
problems to be considered below, the initial states will be fixed and the parameter
vector will result from the parameterization of a control profile $\theta(t)$. The model is
expressed in autonomous form; a non-autonomous system can readily be converted
to autonomous form by treating the independent variable \((t)\) as an additional state
variable. Systems described by DAE models will be considered elsewhere.

The optimization problem to be considered is

\[
\min_{\theta, x_\eta} \phi = \phi[x(t_\eta, \theta), \theta; \eta = 0, 1, \ldots, r]
\]

\[\text{s.t.} \quad 0 \geq g(x, \theta)\]
\[\dot{x} = f(x, \theta)\]
\[x(t_0) = x_0(\theta)\]
\[t \in [t_0, t_r] = [t_0, t_f]\]
\[\theta \in \Theta.\]

The objective \(\phi\) is expressed in terms of the adjustable parameters \(\theta\) and the
state values at discrete points \(t_\eta, \eta = 0, 1, \ldots, r\). These discrete points may be
specified in the problem formulation, or may arise in the numerical solution of the
ODE model. If the objective function involves an integral, it can be eliminated
using an appropriate quadrature variable. The IPCs are given by the function vec-
tor \(g\) (length \(m\)). It is assumed that \(f\) is \((\kappa - 1)\)-times continuously differentiable
with respect to the state variables \(x\), and \((q + 1)\)-times continuously differentiable
with respect to the parameters \(\theta\). It is also assumed that \(\phi\) and \(g\) are \((q + 1)\)-times
continuously differentiable with respect to \(\theta\). Here \(\kappa\) is the order of the trunca-
tion error in the interval Taylor series (ITS) method to be used in the integration
procedure (to be discussed in Section 2.4), and \(q\) is the order of the Taylor model
to be used to represent parameter dependence (to be discussed in Section 2.2).

When the sequential approach is applied, a numerical ODE solver will provide
values of $x_\eta = x(t_\eta, \theta)$, $\eta = 1, \ldots, r$ for specified values of $\theta$, thus effectively eliminating $x_\eta$, $\eta = 0, \ldots, r$ and leaving only the adjustable parameters $\theta$ as decision variables. $\Theta$ is a vector of interval bounds on the decision variables, thus defining the space to be searched for the global optimum. The feasible region, denoted as $G$, is a subset of $\Theta$, given by $G = \{ \theta \in \Theta \mid 0 \geq g(x, \theta), \forall t \in [t_0, t_f] \}$.

We will use a branch-and-bound approach to address the problem stated above. To assure convergence in a finite number of steps, deterministic global optimization algorithms using this approach are typically terminated at a solution point that is within some specified tolerance of the true global minimum. For constrained problems, it is customary to use two such tolerances: 1. The value of the objective function at the solution point may not exceed the true global minimum value by more than some tolerance, usually denoted as $\epsilon$; 2. The solution point may not lie outside of the feasible region by more than some tolerance. That is, the feasible region $G$ is approximated by an outer approximation $G^+ \supset G$ that never differs from $G$ by more than the tolerated amount. Using an outer approximation of the feasible region to determine constraint satisfaction has the advantage that all of $G$ will be searched for the global minimum, but the disadvantage that the final solution point may in fact be outside of $G$. In many contexts, such a small constraint violation is tolerable. However, it is not uncommon for IPCs to be safety related, thus dictating a more conservative approach to constraint satisfaction in this context. We thus choose to approximate the feasible region using an inner approximation $G^- \subset G$ that approaches $G$ to within some tolerance. This has the advantage that the final solution point is guaranteed to be feasible, but the disadvantage that there may be very small regions of $G$ that are not conclusively searched for an $\epsilon$-global minimum. In choosing the inner-approximation approach,
the search for the global minimum is restricted to the space $\mathcal{G}^-$ in which constraint satisfaction is rigorously guaranteed, and we seek to determine an $\epsilon$-global optimum within this space. We will also bound the potential improvement in the global minimum that might occur if the search was expanded to $\mathcal{G}^+$. 

3.3 Global Optimization Algorithm

In this section, we present a new method for the rigorous global optimization of dynamic systems subject to IPCs. As noted previously, when a sequential approach is used, the state variables are effectively eliminated from the objective function using the ODE constraints, leaving a constrained minimization of $\phi(\theta)$ with respect to the adjustable parameters (decision variables) $\theta$. The new global optimization algorithm is constructed on a branch-and-bound framework with domain reduction using constraint propagation with Taylor models. Branch-and-bound with domain reduction is sometimes referred to as a branch-and-reduce approach [98].

As discussed in Section 3.2, our goal is to determine an $\epsilon$-global minimum of $\phi(\theta)$ within an inner approximation $\mathcal{G}^-$ of the feasible region $\mathcal{G}$. Specifically, the feasible region will be approximated, as needed, by an inner subpaving of precision $\delta$. An inner subpaving $\mathcal{G}^-$ is a set of nonoverlapping subintervals, all of which can be rigorously proven to belong to the feasible region $\mathcal{G}$, and which excludes all subintervals that can be proven not to be in $\mathcal{G}$ and all subintervals for which membership in $\mathcal{G}$ cannot be decided. An outer subpaving $\mathcal{G}^+$ contains $\mathcal{G}^-$ and those subintervals for which membership in $\mathcal{G}$ cannot be decided. The subpaving precision $\delta$ is the width of the largest undecided subinterval. As $\delta$ is reduced, $\mathcal{G}^-$ will more closely approach $\mathcal{G}$, with $\delta$ providing a measure, in terms of the decision
variables, of the closeness of approach. The concept of subpavings is described in detail, with several examples, by Jaulin et al. [54]. An entire inner subpaving for the feasible region rarely needs to be generated since many subintervals are eliminated from the search domain based on their objective function values. In the procedure described below, the subpaving precision $\delta$ can be specified directly by the user, or it can be tuned (Section 3.3.5) based on a specified constraint resolution parameter $\omega$. In effect, $\omega$ will provide an approximate measure, in terms of the constraint values, of the closeness with which $G^-$ must approach $G$.

3.3.1 Initialization

In a branch-and-bound procedure, a key issue is how to initialize $\hat{\phi}$, an upper bound on the global minimum. This is often done by evaluating $\phi(\theta)$ at some feasible point $\hat{\theta}$. One simple approach for selecting an appropriate $\hat{\theta}$ is to evaluate $\phi(\theta)$ at one or more feasible $\theta$ points, and to designate the best (lowest $\phi$ value) as $\hat{\theta}$. However, finding and verifying a feasible point may be nontrivial. An alternative approach is to use available local optimization and stochastic global optimization tools to determine a $\hat{\theta}$. In general, we would expect the latter approach to be preferable (this is shown computationally in Section 3.4.1). For this purpose we use the MATLAB toolbox DOTcvp [48], which solves constrained optimal control problems using a sequential approach. This toolbox provides a variety of local, stochastic global, and hybrid optimization methods together with initial value problem (IVP) solvers from SUNDIALS [47] for integration of the ODE constraints. SUNDIALS includes the Adams-Moulton formulas for nonstiff problems and backward differentiation formulas for stiff problems. Gradient information for use in the optimization can also be obtained from SUNDIALS either by
finite difference or by using the sensitivity equations. We use the default method in DOTcvp for optimization, namely FMINCON \([77]\) with a sequential quadratic programming algorithm, thus obtaining a result for \(\hat{\theta}\). It has been our experience, however, that the DOTcvp result may, in some cases, have slight violations of the path constraints (even when using very tight error tolerances in the underlying numerical routines). This is apparently due to the use in DOTcvp of IVP solvers that are not rigorously guaranteed. Such constraint violations can be confirmed using VSPODE, which is a rigorously guaranteed IVP solver. This issue can easily be dealt with by systematically making small changes in the DOTcvp result until its feasibility can be confirmed using VSPODE. To evaluate \(\phi(\hat{\theta})\) rigorously, we use a procedure described in Section 3.3.3 that results in interval bounds on \(\phi\). The upper bound of this interval is then used as the initial \(\hat{\phi}\).

An additional initialization step is to establish a work list (stack) \(\mathcal{L}\). The work list \(\mathcal{L}\) will contain a sequence of subintervals (boxes) that need to be tested, as described below, and initially \(\mathcal{L} = \{\Theta\}\), the entire specified parameter (decision variable) space. Each subinterval in \(\mathcal{L}\) has a status flag that can have a value of TRUE, indicating a box that is rigorously guaranteed to satisfy the IPCs and may contain an \(\epsilon\)-global minimum, a value of UNDECIDED, indicating a box that may satisfy the IPCs and may contain an \(\epsilon\)-global minimum, or a value of FALSE, indicating that the box is rigorously guaranteed either not to satisfy the IPCs or not to contain an \(\epsilon\)-global minimum. Once a box is marked FALSE it can be removed from \(\mathcal{L}\) and is no longer tested. The initial element of \(\mathcal{L}\) is marked UNDECIDED.

We also establish a list \(\mathcal{L}^+\), initially empty, which will contain boxes in which testing for an \(\epsilon\)-global minimum is terminated at an incomplete stage, together
with objective function bounds over each such box. These are boxes of width less than the subpaving tolerance $\delta$ that belong to the outer subpaving $G^+$, but not the inner subpaving $G^-$ within which an $\epsilon$-global minimum is sought. By saving these boxes, we can later establish a bound on the improvement in $\phi$ that is possible if $G^+$ is used to approximate the feasible region instead of $G^-$. Finally, we set the convergence tolerance $\epsilon$ for the objective function, the subpaving precision $\delta$ for the feasible region, and the parameters used by VSPODE, namely the stepsize $h$, the ITS truncation order $\kappa$ and the Taylor model order $q$. The subpaving precision and VSPODE parameters can be set by using an optional “autotuning” procedure based on the (user-specified) IPC resolution parameter $\omega$. We will defer explanation of the optional autotuning procedure until after the details of the optimization algorithm are presented.

3.3.2 Rigorous IPC Test

At the $k$-th iteration of the overall algorithm (to be discussed in Section 3.3.4), a subinterval is removed from the front of $L$ for testing. This subinterval is denoted $\Theta^{(k)}$. The purpose of the test procedure described in this section is to determine rigorously whether all $\theta \in \Theta^{(k)}$ lead to states that satisfy the IPCs for all $t \in [t_0, t_f]$, and, if not, to try to shrink $\Theta^{(k)}$ by using constraint propagation. Only boxes flagged as UNDECIDED undergo this test. If a box is already marked TRUE, then it has already been proven that all $\theta \in \Theta^{(k)}$ lead to states that satisfy the IPCs for all $t \in [t_0, t_f]$ and so this test need not be performed again.

Since we will assume that the specified initial states satisfy the IPCs, the subinterval $\Theta^{(k)}$ is first given a status of TRUE as of $t = t_0$. We then begin carrying out integration time steps using information obtained from VSPODE. Consider
the time step beginning at \( t = t_j \). Recall that in the first phase (verification phase) of VSPODE, rigorous interval bounds \( \tilde{X}^{(k)}_j \) on the state variables can be obtained that are valid for all \( \theta \in \Theta^{(k)} \) and for all \( t \in [t_j, t_{j+1}] \), that is, for the entire integration time step, not just the endpoint \( t_{j+1} \). Then, using interval arithmetic, the corresponding bounds \( \tilde{G}^{(k)}_j = g(\tilde{X}^{(k)}_j, \Theta^{(k)}) \) on the IPCs can be computed, and these are also valid over the entire time step. However, the bounds \( \tilde{X}^{(k)}_j \) are often only relatively loose bounds, which may lead to poor performance in bounding the IPCs. To improve \( \tilde{X}^{(k)}_j \), we will use an idea suggested by Lin and Stadtherr [65], based on the observation that over a small time step it is likely that most, if not all, of the state trajectories will increase or decrease monotonically with time. This leads to the following procedure:

1. Compute \( \tilde{X}^{(k)}_j \) using the first phase of VSPODE. If the verification step fails (eq. (2.34) cannot be satisfied) then integration with VSPODE cannot continue; thus \( \Theta^{(k)} \) is marked as UNDECIDED and the IPC test terminates (this is only likely to occur in early iterations, when \( \Theta^{(k)} \) may be relatively large). Otherwise continue with computation of \( X^{(k)}_{j+1} \) using the second phase of VSPODE.

2. For each variable \( i = 1, \ldots, n \), compute bounds on \( \dot{x}_i = f_i(x, \theta) \) over the current time step by determining \( F_i^{(k)} = f_i(\tilde{X}^{(k)}_j, \Theta^{(k)}) \) using interval arithmetic. If \( F_i^{(k)} \geq 0 \) or \( F_i^{(k)} \leq 0 \) (\( x_i \) monotonically increases or decreases with respect to time), then the interval hull of \( X^{(k)}_{j,i} \) and \( X^{(k)}_{j+1,i} \) can be taken as the improved (tightened) \( \tilde{X}^{(k)}_{j,i} \). This is done component by component until the entire improved \( \tilde{X}^{(k)}_j \) is obtained.

This improved \( \tilde{X}^{(k)}_j \) is now used to compute the IPC bounds \( \tilde{G}^{(k)}_j \). By examining \( \tilde{G}^{(k)}_j, \Theta^{(k)} \) can be put into one of three categories:
1. If, for some IPC \( l \), \( \widetilde{G}^{(k)}_{j,l} > 0 \), then this is rigorous proof that the IPC \( g_l \leq 0 \) cannot be satisfied by any point \( \theta \in \Theta^{(k)} \). Thus, we can flag \( \Theta^{(k)} \) as FALSE and stop the IPC test.

2. If \( \widetilde{G}^{(k)}_j \leq 0 \) and \( \Theta^{(k)} \) was TRUE at time \( t_j \), then this is rigorous proof that, up to time \( t_{j+1} \), all \( \theta \in \Theta^{(k)} \) satisfy all of the constraints. Thus, we can flag \( \Theta^{(k)} \) as TRUE as of time \( t_{j+1} \), and proceed to the next integration time step. Note that a box maintains TRUE status only if constraint satisfaction is rigorously guaranteed for all \( \theta \in \Theta^{(k)} \) for all previous time steps in addition to the current one; that is, constraint satisfaction is guaranteed for all \( t \in [t_0, t_{j+1}] \).

3. Otherwise, we flag \( \Theta^{(k)} \) as UNDECIDED, and proceed with additional testing based on information obtained in the second phase of VSPODE.

From the second phase of VSPODE, we can obtain a Taylor model \( T^{(k)}_{x_{j+1}}(\theta) \) giving the states \( x_{j+1} \) at \( t_{j+1} \) in terms of the parameters (decision variables) \( \theta \). Using Taylor model arithmetic, we can then compute \( T^{(k)}_{g_{j+1}}(\theta) = g_{j+1}(T^{(k)}_{x_{j+1}}, T_{\theta}) \), a Taylor model of the IPCs at \( t_{j+1} \) in terms of \( \theta \). By now bounding \( T^{(k)}_{g_{j+1}}(\theta) \) over \( \theta \in \Theta^{(k)} \), we obtain interval bounds \( G^{(k)}_{j+1} \) on the IPCs at \( t_{j+1} \). These bounds may be significantly tighter than \( \widetilde{G}^{(k)}_j \), but are valid only at \( t = t_{j+1} \) (\( \widetilde{G}^{(k)}_j \) gives bounds valid for \( t \in [t_j, t_{j+1}] \)). Based on this information, there are three cases to consider:

1. If for some IPC \( l \), \( G^{(k)}_{j+1,l} > 0 \), then this is rigorous proof that at \( t = t_{j+1} \) the constraint \( g_l \leq 0 \) is violated at all points \( \theta \in \Theta^{(k)} \). Thus, we can flag \( \Theta^{(k)} \) as FALSE and stop the IPC test.

2. If \( G^{(k)}_{j+1} \leq 0 \), this shows that all the constraints hold for \( \theta \in \Theta^{(k)} \) at \( t = t_{j+1} \). But since we require constraint satisfaction at all points in time, not just
at the discrete time-step endpoints, we will leave the UNDECIDED flag on \( \Theta^{(k)} \), and continue to the next integration time step.

3. Otherwise, use the IPC Taylor model \( T_{g_{j+1}}^{(k)}(\boldsymbol{\theta}) \) in a constraint propagation procedure (CPP) based on the constraint \( g_{j+1} \leq 0 \), as discussed in Section 2.3. The CPP will attempt to shrink, or perhaps completely eliminate, \( \Theta^{(k)} \) by removing regions from it for which it is impossible to satisfy the IPCs. If \( \Theta^{(k)} \) is completely eliminated then it is marked FALSE and the IPC test terminates. Otherwise, the updated (possibly smaller) \( \Theta^{(k)} \) retains the UNDECIDED flag, and we continue to the next integration time step.

If \( T_{g_{j+1}}^{(k)}(\boldsymbol{\theta}) \) depends on only some of the components of \( \boldsymbol{\theta} \), then this can be used to reduce the computational effort needed to apply the CPP.

The IPC test procedure described above is applied to \( \Theta^{(k)} \) beginning with \( t_j = t_0 \). Possible overall outcomes are:

1. Integration terminates early due to failure in the verification phase of VSPODE.

   In this case, \( \Theta^{(k)} \) will have a status of UNDECIDED, and will be bisected upon return to the main algorithm.

2. Integration terminates early due to proof that no \( \boldsymbol{\theta} \in \Theta^{(k)} \) can satisfy the IPCs. In this case, \( \Theta^{(k)} \) will have a status of FALSE and will not be further tested. On return to the main algorithm, a new subinterval will be taken from the front of \( \mathcal{L} \) for testing.

3. Integration terminates at the specified final time \( t_f \) and \( \Theta^{(k)} \) has a status of TRUE. This is rigorous proof that all the IPCs are satisfied for all \( \boldsymbol{\theta} \in \Theta^{(k)} \) and for all \( t \in [t_0, t_f] \) (not just for discrete points between \( t_0 \) and \( t_f \)). On return to the main algorithm this box will be sent to the objective test.
4. Integration terminates at the specified final time $t_f$ and $\Theta^{(k)}$ has a status of UNDECIDED. In this case, $\Theta^{(k)}$ may contain some values of $\theta$ that satisfy the IPCs and some that do not. It is also possible that all $\theta \in \Theta^{(k)}$ satisfy the constraints, or that all $\theta \in \Theta^{(k)}$ do not satisfy the constraints, but that this could not be rigorously verified due to the overestimation of bounds. On return to the main algorithm this box will be sent to the objective test.

3.3.3 Objective Test

The purpose of the test procedure described in this section is to determine rigorously whether the tested interval $\Theta^{(k)}$ contains any points $\theta$ at which $\hat{\phi}$, the current upper bound on the global minimum, may be improved by more than the tolerance $\epsilon$. If possible, we will also seek to improve the value of $\hat{\phi}$ and to shrink $\Theta^{(k)}$ by using constraint propagation.

By running VSPODE we can obtain Taylor models of the state variables over $\Theta^{(k)}$ at all times needed to evaluate the objective function (if possible we will use VSPODE results obtained during the IPC test). Using Taylor model arithmetic, we can then compute $T_{\hat{\phi}}^{(k)}(\theta)$, a Taylor model over $\Theta^{(k)}$ of the objective function in terms of the decision variables $\theta$. By now bounding $T_{\hat{\phi}}^{(k)}(\theta)$ over $\theta \in \Theta^{(k)}$, we obtain interval bounds $\Phi^{(k)}$ on the objective function over $\Theta^{(k)}$. Based on these bounds, we proceed as follows:

1. If $(\hat{\phi} - \Phi^{(k)}) \leq \epsilon$ then the current $\Theta^{(k)}$ does not contain any points at which $\hat{\phi}$ can be improved by more than the tolerance $\epsilon$. Thus, $\Theta^{(k)}$ is marked as FALSE and the objective test is stopped.

2. If $\Theta^{(k)}$ is TRUE at this point, then it may contain feasible points that can be used to improve $\hat{\phi}$ by more than $\epsilon$. Therefore, we apply a local optimization
procedure over $\Theta^{(k)}$ to try to determine a better $\hat{\theta}$. For this purpose, we use the bound-constrained quasi-Newton method L-BFGS-B [17] for local optimization, and DDASSL [13] as the integration routine to be called by L-BFGS-B. If a better $\hat{\theta}$ is found, then we proceed to evaluate $\hat{\phi}$ rigorously at the new $\hat{\theta}$. To do this, we run VSPODE on the degenerate interval input $\hat{\Theta} = [\hat{\theta}, \hat{\theta}]$ and use the results to determine a Taylor model for $\phi$ over $\hat{\Theta}$. Bounding this Taylor model yields the interval $\hat{\Phi}$ (this is nonzero width because it bounds truncation error in the ODE solver and because of outward rounding in the bounding process). Then, if $\hat{\Phi} < \hat{\phi}$, a new upper bound on the global minimum is set as $\hat{\phi} = \hat{\Phi}$. Since $\Phi^{(k)}$ usually will be somewhat less than the true lower bound on $\phi$ over $\Theta^{(k)}$, performing a local optimization in a TRUE box whenever $\hat{\phi} - \Phi^{(k)} > \epsilon$, as done here, will lead to some unsuccessful attempts to improve $\hat{\phi}$. An alternative approach is to perform a local optimization only when $\Phi^{(k)} < \hat{\phi}$, which guarantees that an improved $\hat{\phi}$ will be found. While the latter approach avoids futile attempts to improve $\hat{\phi}$, the tradeoff is that the former approach is likely to find improved $\hat{\phi}$ values more often, potentially leading to the elimination of some subintervals at an earlier point in the branch-and-bound process. At least for the example problems considered here, the former approach is usually somewhat more efficient.

3. Use the objective function Taylor model $T^{(k)}_{\phi}(\theta)$ in a CPP based on the constraint $\hat{\phi} - \hat{\Phi} + \epsilon \leq 0$, which must be satisfied for any point $\theta$ to be a candidate for the global minimum. The CPP will try to reduce $\Theta^{(k)}$ by removing regions from it for which it is not possible to satisfy this constraint. It is also possible, since the CPP involves a recursive process in
which the $\Phi^{(k)}$ interval is tightened as $\Theta^{(k)}$ is reduced, for the entire $\Theta^{(k)}$ to be eliminated. In this case, $\Theta^{(k)}$ is marked FALSE.

For a tested subinterval $\Theta^{(k)}$, the potential overall outcomes of the objective test are:

1. The subinterval is marked FALSE because it has been shown rigorously to contain no points for which the possibly updated $\hat{\phi}$ can be improved by more than the tolerance $\epsilon$.

2. The subinterval has been reduced in size without change in status (still either TRUE or UNDECIDED).

3. No change in the subinterval or its status has occurred.

3.3.4 Overall Optimization Algorithm

The overall global optimization algorithm proceeds as follows:

1. Initialize (Section 3.3.1).

2. Set iteration counter $k = 0$ and (optional) perform automatic tuning (Section 3.3.5).

3. Remove from the front of $L$ a subinterval $\Theta^{(k)}$ for testing. This subinterval may have status of either UNDECIDED or TRUE.

4. If $\Theta^{(k)}$ has status UNDECIDED, then send it to the IPC test (Section 3.3.2). Otherwise, run VSPODE and proceed directly to step 5.

   (a) If the IPC test terminates due to failure of the verification phase in VSPODE, go to step 6 (bisection).
(b) If $\Theta^{(k)}$ returns from the IPC test with status FALSE, go to step 7.

(c) Otherwise, proceed to step 5.

5. Send $\Theta^{(k)}$ to the objective test (Section 3.3.3).

   (a) If $\Theta^{(k)}$ returns from the objective test with status FALSE, go to step 7.

   (b) If $\Theta^{(k)}$ returns from the objective test with status UNDECIDED and its width is less than the subpaving precision $\delta$, then $\Theta^{(k)}$ belongs to the outer approximation (subpaving) $G^+$ but not to the inner approximation (subpaving) $G^-$ in which an $\epsilon$-global minimum is sought. Thus, no further testing of $\Theta^{(k)}$ is done, and $\Theta^{(k)}$ and $\Phi^{(k)}$ are added together to the list $L^+$ of boxes belonging to $G^+$ in which testing for an $\epsilon$-global minimum was incomplete. Go to step 7.

   (c) If $\Theta^{(k)}$ returns from the objective test with its volume reduced by 70% or more (compared to its volume at the beginning of step 4), then put the reduced $\Theta^{(k)}$ at the front of $L$, increment $k$ by one, and go to step 3. The threshold of 70% volume reduction is a heuristic and may be adjusted if desired.

   (d) Otherwise, proceed to step 6.

6. Bisect $\Theta^{(k)}$ using the component with the largest relative width. The resulting subintervals retain the status flag of the parent subinterval $\Theta^{(k)}$ and are placed at the front of $L$. Increment $k$ by one and go to step 3.

7. If $L$ is empty, proceed to step 8. Otherwise, increment $k$ by one and return to step 3.
8. Examine objective bounds saved in $\mathcal{L}^+$ to determine the overall lower bound

$$\phi^+ = \min_{\Phi^{(i)} \in \mathcal{L}^+} \Phi^{(i)}$$
on $\phi$ over boxes in $\mathcal{L}^+$. Terminate.

A flowchart of the overall algorithm is given in Figure 3.1. At the termination of this procedure, $\mathcal{L}$ is empty and the final $\hat{\phi}$ is an $\epsilon$-global minimum $\phi^*$ of the dynamic optimization problem subject to IPCs. The corresponding $\epsilon$-global minimizer $\theta^*$ is the final $\hat{\theta}$. In step 8, we analyze the list $\mathcal{L}^+$ of boxes, saved in step 5(b), not completely tested for an $\epsilon$-global minimum because they belong to $\mathcal{G}^+$ but not to $\mathcal{G}^−$. Interval bounds on $\phi$ in each of these boxes have been generated previously in the objective test and saved in step 5(b). These bounds can be used to establish a lower bound $\phi^+$ on the $\phi$ values possible over all boxes in $\mathcal{L}^+$. This provides a bound, $\phi^* - \phi^+ = \epsilon^+$ on the objective function improvement that is possible if $\mathcal{G}^+$ is used to approximate the feasible region instead of $\mathcal{G}^−$. It is possible, if the boxes in $\mathcal{L}^+$ were put there before the final $\hat{\phi}$ value is reached, that $\epsilon^+ \leq \epsilon$, meaning that $\phi^*$ is an $\epsilon$-global minimum in both $\mathcal{G}^−$ and $\mathcal{G}^+$. If $\epsilon^+ > \epsilon$, then $\phi^*$ is an $\epsilon$-global minimum in $\mathcal{G}^−$ and an $\epsilon^+$-global minimum in $\mathcal{G}^+$. Should the global minimum in $\mathcal{G}^+$ be of interest (for reasons explained previously we are interested primarily in $\mathcal{G}^−$) and $\epsilon^+$ is unacceptably large, then the appropriate boxes in $\mathcal{L}^+$ can be further processed, by applying the algorithm with smaller values of $\delta$ and using higher precision (smaller $h$) in VSPODE.

Note that the algorithm is constructed in a way that intertwines testing for IPC satisfaction and $\epsilon$-global optimality. This is much more efficient than an approach in which one first identifies the domain that satisfies the IPCs and then searches that domain for the global optimum. By intertwining the use of the IPC test and the objective test we greatly reduce the need for state bounding and eliminate much redundant work. The algorithm uses a standard depth-first search

45
approach. It is possible that a different approach for ordering the search process could be more efficient.

3.3.5 Tuning Procedure

The parameters that must be set before executing the algorithm described above are the convergence tolerance \( \epsilon \) for the optimization problem, the stepsize \( h \), the ITS truncation order \( \kappa \) and Taylor model order \( q \) used in VSPODE, and the subpaving precision \( \delta \). In this subsection we will describe an optional procedure for automatic tuning of \( h, \kappa, q \) and \( \delta \), based on user-specified values of \( \epsilon \) and of \( \omega \), a constraint resolution parameter. This constraint resolution parameter is used only for the purpose of this autotuning procedure. The goal is to find a stepsize \( h \) and subpaving precision \( \delta \) such that the smallest boxes investigated in the optimization algorithm are likely to cover an interval of IPC values within the resolution \( \omega \) and an interval of objective function values within the tolerance \( \epsilon \). Also, to reduce the computational cost, we try to reduce \( \kappa \) and \( q \) (from their heuristic default values of \( \kappa = 9 \) and \( q = 5 \)), while maintaining satisfaction of the \( \omega \) and \( \epsilon \) conditions. For doing all this, we use the heuristic procedure outlined below.

The procedure used is based on a random sample of \( \sigma \) points \( \theta_s, s = 1, \ldots, \sigma \), from the search domain \( \Theta \). By adjusting the number of sample points, the user can decide whether to invest more or less computational effort in this tuning process. Because the computational cost of finding a rigorous \( \epsilon \)-global minimum can become quite large as the number of decision variables increases, we are willing in general to devote a significant computational effort to achieve a reasonable tuning. However, in our experience on the example problems considered below, a
relatively small sample of $\sigma = 5$ points is adequate.

In the tuning procedure, first $h$ is adjusted, then $\delta$ and finally $\kappa$ and $q$, as follows:

1. (Stepsize) As discussed in Section 2.4.3, because of the dependency problem of interval arithmetic, using too large a value of the stepsize $h$ can lead to excessive overestimation of the state bounds and thus of the IPC bounds. We would like to choose $h$ so that, on average, the width of the IPC bounds (including overestimation) does not exceed the specified IPC resolution parameter $\omega$. To do this we will use an iterative process in which VSPODE is run at each iteration using each sampled $\theta_s$ value, treated as a zero-width interval. By using zero-width parameter intervals we are able to focus on the effect of $h$ alone on overestimation. At the $u$-th iteration the time step will be $h^{(u)}$ and the total number of time steps will be $\tau^{(u)} = (t_f - t_0)/h^{(u)}$ (assumed to be an integer). After each iteration, the stepsize is reduced by a factor of 10, so $h^{(u+1)} = h^{(u)}/10$ and $\tau^{(u+1)} = 10\tau^{(u)}$.

The iterative process is:

(a) Initialize iteration counter $u = 0$ and set $h^{(u)}$ (user specified).

(b) Run VSPODE with fixed stepsize $h = h^{(u)}$ for each sampled point $\theta_s$, $s = 1, \ldots, \sigma$, thus obtaining for the time step beginning at $t_j$, $j = 0, \ldots, \tau^{(u)} - 1$ the state bounds $\widetilde{X}_{j,s}^{(u)}$. If verification failure occurs in phase one of VSPODE at any time step, then go to (f).

(c) Compute the corresponding IPC bounds $\widetilde{G}_{j,s}^{(u)} = g(\widetilde{X}_{j,s}^{(u)}, \theta_s)$ and deter-
mine the average IPC width (over all samples and time steps)

\[ \hat{w}(\tilde{G}^{(u)}) = \frac{\sum_{s=1}^{\sigma} \sum_{j=0}^{\tau(u)-1} w(\tilde{G}_{j,s}^{(u)})}{\sigma \tau^{(u)}} \]  

(3.2)

(d) If \( \hat{w}(\tilde{G}^{(u)}) \leq \omega \), stop iterating and accept the current stepsize \( h^{(u)} \) as \( h \).

(e) If \( u = 0 \), go to (f). Otherwise we compare the average IPC width at the current iteration with the average IPC width over the same time interval during the previous iteration. This is done to provide a measure of the sensitivity of the average IPC width to reduction in the stepsize. The time interval of one time step in the previous iteration corresponds to ten time steps in the current iteration. Thus we must take the IPC bounds in the current iteration and unite them ten at a time, corresponding to the time steps in the previous iteration. This can be expressed by

\[ \tilde{\Gamma}_{l,s}^{(u)} = \bigcup_{j=10l}^{10l+9} \tilde{G}^{(u)}_{j,s} \]  

(3.3)

where \( l = 0, \ldots, \tau^{(u-1)} - 1 \). Then the average IPC width for comparison to the previous iteration is

\[ \hat{w}(\tilde{\Gamma}^{(u)}) = \frac{\sum_{s=1}^{\sigma} \sum_{l=0}^{\tau^{(u-1)}-1} w(\tilde{\Gamma}_{l,s}^{(u)})}{\sigma \tau^{(u-1)}}. \]  

(3.4)

If there has been relatively little improvement in the average IPC width,
say less than half of the tolerance $\omega$,

$$\hat{w}(\tilde{G}^{(u-1)}) - \hat{w}(\tilde{T}^{(u)}) \leq \frac{\omega}{2}, \quad (3.5)$$

then we stop iterating and use the stepsize $h^{(u-1)}$ from the previous iteration as $h$, since using the larger stepsize will save computational effort with relatively little difference in IPC bound quality. If there has been somewhat greater improvement in the average IPC width, say

$$\frac{\omega}{2} \leq \hat{w}(\tilde{G}^{(u-1)}) - \hat{w}(\tilde{T}^{(u)}) \leq \omega, \quad (3.6)$$

then stop iterating and accept the current stepsize $h^{(u)}$ as $h$. Otherwise, continue.

(f) Set $h^{(u+1)} = h^{(u)}/10$ and $\tau^{(u+1)} = 10\tau^{(u)}$.

(g) Increment $u$ by one and return to (b).

2. (Subpaving precision) In the second tuning step we adjust the subpaving precision $\delta$. Here we try to account for overestimation of the state bounds, and thus the objective and IPC bounds, due to parameter interval width (see Section 2.4.3). We would like to choose $\delta$ to be sufficiently small so that, on average, the width of the objective interval (including overestimation) does not exceed the objective tolerance $\epsilon$ and the width of the IPC interval (including overestimation) does not exceed the IPC resolution parameter $\omega$. To do this we will use an iterative process in which VSPODE is run at each iteration using a small parameter box, centered on each sampled $\theta_s$ value, $s = 1, \ldots, \sigma$. At the $u$-th iteration the small box width will be $\delta^{(u)}$. The small parameter boxes used are then $\Theta^{(u)}_s = \theta_s + [-\delta^{(u)}/2, \delta^{(u)}/2]$,
s = 1, . . . , σ. After each iteration the small box width will be reduced by a factor of 10, so δ\textsuperscript{(u+1)} = δ\textsuperscript{(u)}/10. The stepsize h used is the value determined in the previous tuning step, and the corresponding number of time steps is τ. The iterative process is:

(a) Initialize iteration counter \( u = 0 \) and set \( δ\textsuperscript{(u)} \) (user specified).

(b) Run VSPODE with fixed stepsize \( h \) for each small parameter box \( Θ_s^{(u)} \), \( s = 1, . . . , σ \), thus obtaining for the time step beginning at \( t_j, j = 0, . . . , τ − 1 \) the state bounds \( \tilde{X}_{j,s}^{(u)} \). If verification failure occurs in phase one of VSPODE at any time step, then go to (e).

(c) Based on these state bounds compute the corresponding IPC bounds \( \tilde{G}_{j,s}^{(u)} \) and objective bounds \( Φ_s^{(u)} \) using interval arithmetic. Determine the average IPC width \( \hat{w}(\tilde{G}^{(u)}) \) using eq. (3.2) (with \( τ^{(u)} = τ \)), and the average objective width from

\[
\hat{w}(Φ^{(u)}) = \frac{\sum_{s=1}^{σ} w(Φ_s^{(u)})}{σ}.
\]

(d) If \( \hat{w}(Φ^{(u)}) ≤ ϵ \) and \( \hat{w}(\tilde{G}^{(u)}) ≤ ω \), stop iterating and accept the current small box tolerance \( δ^{(u)} \) as \( δ \). Otherwise, continue.

(e) Set \( δ^{(u+1)} = δ^{(u)}/10 \).

(f) Increment \( u \) by one and return to (b).

3. (ITS truncation order and Taylor model order) Finally, to reduce the computational expense in running VSPODE, we try to reduce the ITS truncation order \( κ \) and the Taylor model order \( q \), while still maintaining satisfaction of the conditions used to establish the values for \( h \) and \( δ \) in the previous
two tuning steps. We start by reducing to $\kappa = 5$ and $q = 3$, the minimum values that, in our experience [63], lead to good performance. For these trial values of $\kappa$ and $q$, we then repeat tuning steps 1 and 2 and check whether the criteria for accepting the established $h$ and $\delta$ are still met. If so, the trial values of $\kappa$ and $q$ are accepted. Otherwise we systematically increase first $q$ and then $\kappa$ until these criteria are satisfied.

There certainly may be other ways to tune the algorithm settings. We have found this heuristic procedure useful to identify and control, before actually attempting the global optimization procedure, the impact of bound overestimation on a specific optimization problem. The autotuning procedure described is relatively expensive computationally, but is still reasonable in this context since the cost of finding a rigorous $\epsilon$-global minimum may become quite large. While this tuning procedure can be considered optional, we recommend its use. In the next section, we will study the computational performance of the $\epsilon$-global optimization algorithm described here.
Figure 3.1. Overall algorithm flowchart
3.4 Computational Studies

In this section, we apply the algorithm outlined above to three semi-batch reactor problems, each adapted from systems studied by Srinivasan et al. [107]. We use these examples to demonstrate the effectiveness of the approach proposed above, as well as to study various aspects of the algorithm. In all cases, there is an active IPC at the globally optimal solution. The first example involves feed rate optimization for a second-order exothermic reaction in an isothermal semi-batch reactor. We also use this problem to consider the effect of the initialization for $\hat{\varphi}$. The second problem involves temperature optimization for an exothermic series reaction in a nonisothermal semi-batch reactor. This problem is also used to consider the effects of the IPC resolution parameter $\omega$ and stepsize parameter $h$. The third problem considers feed rate optimization for parallel reactions in an isothermal semi-batch reactor. This example is also used to study the effect of using different values for the ITS truncation order $\kappa$ and Taylor model order $q$. All the example problems were solved on a single 2.6 GHz dual-core AMD Opteron CPU running Red Hat Linux. All reported CPU times are rounded to three significant digits. The entire algorithm was implemented in C++.

3.4.1 Example 3.1: Second-order Exothermic Reaction in an Isothermal Semi-batch Reactor

In this section, we consider a feed rate optimization problem involving a second-order exothermic reaction in an isothermal semi-batch reactor. There are two IPCs, one a cooling-failure safety constraint and the other a maximum volume constraint. We solved two versions of the problem: the first with the safety constraint only and the second with both the safety and volume constraints.
Consider the reaction
\[ A + B \xrightarrow{k} C \] (3.8)

occurring in an isothermal semi-batch reactor fed a stream containing B, and operating at temperature \( T \). For the problem specification considered here, B is the limiting reactant. The objective is to maximize the amount of product C at the final time \( t_f \) by manipulating the volumetric flow rate \( \theta(t) \) of the feed stream, which contains B at concentration \( x_{B,in} \). The problem is formulated as follows:

\[
\begin{align*}
\min_{\theta(t)} \quad & \phi = x_A(t_f)V(t_f) - x_{A0}V_0 \\
\text{IPC(a)} \quad & T_{\text{fail}}(t) = T + x_B(t) \left( \frac{-\Delta H}{\rho c_p} \right) \leq T_{\text{max}} \\
\text{IPC(b)} \quad & V(t) \leq V_{\text{max}} \\
\text{s.t.} \quad & \dot{x}_A = -k x_A x_B - \frac{\theta}{V} x_A \\
& \dot{x}_B = -k x_A x_B + \frac{\theta}{V} (x_{B,in} - x_B) \\
& \dot{V} = \theta \\
& \theta \in [0, 30] \text{ mL/h} = [0, 0.03] \text{ L/h} \\
& t \in [t_0, t_f] = [0, 20] \text{ h}.
\end{align*}
\] (3.9)

Here the state variables are \( x_A(t) \) and \( x_B(t) \), the concentrations of A and B, respectively, and the volume \( V(t) \). Other model parameters are defined, and values given, in Table 3.1 which also gives values of the initial states. In IPC(a), \( T_{\text{fail}} \) is the temperature that could be reached in the case of a cooling failure (assuming an adiabatic operation after failure). If a cooling failure were to occur, the feed of B could be stopped immediately but the B already in the reactor would
continue to react, thus raising the temperature. To guarantee safe operation, $T_{\text{fail}}$ must not be allowed to exceed a specified $T_{\text{max}}$.

### TABLE 3.1

MODEL PARAMETERS FOR EXAMPLE 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Reaction rate constant</td>
<td>0.0482</td>
<td>L/(mol h)</td>
</tr>
<tr>
<td>$x_{A0}$</td>
<td>Initial concentration of A</td>
<td>2</td>
<td>mol/L</td>
</tr>
<tr>
<td>$x_{B0}$</td>
<td>Initial concentration of B</td>
<td>0.5</td>
<td>mol/L</td>
</tr>
<tr>
<td>$V_0$</td>
<td>Initial volume</td>
<td>0.7</td>
<td>L</td>
</tr>
<tr>
<td>$V_{\text{max}}$</td>
<td>Maximum volume</td>
<td>1.1</td>
<td>L</td>
</tr>
<tr>
<td>$x_{B,\text{in}}$</td>
<td>Inlet concentration of B</td>
<td>2</td>
<td>mol/L</td>
</tr>
<tr>
<td>$T$</td>
<td>Reactor temperature</td>
<td>343.15</td>
<td>K</td>
</tr>
<tr>
<td>$T_{\text{max}}$</td>
<td>Maximum allowable temperature</td>
<td>353.15</td>
<td>K</td>
</tr>
<tr>
<td>$c_p$</td>
<td>Heat capacity</td>
<td>4.2</td>
<td>J/(g K)</td>
</tr>
<tr>
<td>$\Delta H$</td>
<td>Heat of reaction</td>
<td>$-60000$</td>
<td>J/mol</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Density</td>
<td>900</td>
<td>g/L</td>
</tr>
</tbody>
</table>

Following the sequential (control parameterization) approach, the flow rate $\theta(t)$ was parameterized as a piecewise constant profile with $p$ equal time intervals.
For example, if \( p = 3 \), then there are three decision variables, \( \theta_1, \theta_2, \) and \( \theta_3 \), representing the flow rates over the time intervals \([0, t_f/3]\), \([t_f/3, 2t_f/3]\), and \([2t_f/3, t_f]\), respectively. For both versions of the problem considered, we solved several different optimization problems, corresponding to different values of \( p \) (from \( p = 1 \) to \( p = 7 \)). All problems were solved to an absolute objective tolerance of \( \epsilon = 10^{-4} \) mol, using an IPC resolution of \( \omega = 10^{-3} \) [K for IPC(a); L for IPC(b)]. Based on the tuning procedure described above, we used \( \delta = 10^{-6} \) L/h, \( h = 0.1 \) h, \( \kappa = 5 \) and \( q = 3 \).

For initializing \( \hat{\phi} \), an upper bound on the global minimum, the two approaches noted in section 3.3.1 were used. In one approach, we randomly sampled points from the search space until a feasible point was found. The value of \( \phi \) at that first feasible point was then used as \( \hat{\phi} \). In the other approach, we obtained \( \hat{\phi} \) using the local solver DOTcvp, with small adjustments of the DOTcvp result as needed to guarantee evaluation of \( \phi \) at a feasible point (see section 3.3.1).

We first considered the version of this problem in which only the safety constraint is imposed. Results for the \( \epsilon \)-global minimum \( \phi^* \) (rounded to the nearest \( 10^{-5} \) mol) and the corresponding \( \epsilon \)-global minimizer \( \theta^* \) (rounded to the nearest \( 10^{-6} \) L/h) are shown in Table 3.2 for the case of initialization by random sampling and in Table 3.3 for the case of initialization by the local solver. As expected, it is clearly more efficient to initialize by using the local solver; for this case we solved problems as large as \( p = 7 \). For these problems the local solver always provided a better (smaller) \( \hat{\phi} \) than random sampling, thus allowing earlier elimination of more of the search domain. However, there is no guarantee that this will always be the case, as a local solver could converge to a poor local minimum. Results of the optimization from the different initializations differ slightly, but are the same
within the absolute tolerance of $\epsilon = 10^{-4}$ mol. Such differences arise because use of different initial $\hat{\phi}$ values will result in the search domain being processed through bisection and constraint propagation in a different order.

We then considered the version of this problem in which both the safety and volume IPCs are imposed. The results obtained are shown in Tables 3.4 and 3.5 for the two different initialization methods used, and are similar, in terms of trends in computational performance, to those seen for the case of the single IPC. The presence of the additional constraint has little impact on the computational cost, which clearly depends strongly on $p$, the number of decision variables. The eventual worst-case exponential complexity with respect to $p$ seen in these results, as well as those in Tables 3.2 and 3.3, reflects the fact that rigorous global optimization for nonlinear problems is in general an NP-hard problem.

TABLE 3.2

RESULTS FOR EXAMPLE 3.1 WITH SAFETY CONSTRAINT ONLY AND INITIALIZATION OF $\hat{\phi}$ WITH RANDOM SAMPLING

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\phi^*$ (mol)</th>
<th>$\theta_1^*$ (L/h) $\times 10^2$</th>
<th>CPU time (s)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-0.58445$</td>
<td>$(2.4648)$</td>
<td>3.35</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>$-0.60786$</td>
<td>$(3.0000, 2.1002)$</td>
<td>3.60</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>$-0.60930$</td>
<td>$(3.0000, 2.7349, 1.9238)$</td>
<td>906</td>
<td>8183</td>
</tr>
<tr>
<td>4</td>
<td>$-0.61060$</td>
<td>$(3.0000, 3.0000, 2.3578, 1.8789)$</td>
<td>1370</td>
<td>10353</td>
</tr>
</tbody>
</table>
$$\begin{array}{cccccc}
 p & \phi^* \text{ (mol)} & \theta^* \text{ (L/h)} \times 10^2 & \text{CPU time (s)} & \text{Iterations} \\
 1 & -0.58444 & (2.4647) & 0.87 & 9 \\
 2 & -0.60785 & (3.0000, 2.0997) & 1.81 & 17 \\
 3 & -0.60934 & (3.0000, 2.7349, 1.9279) & 21.9 & 195 \\
 4 & -0.61055 & (3.0000, 3.0000, 2.3578, 1.8706) & 37.9 & 286 \\
 5 & -0.61128 & (3.0000, 3.0000, 2.8649, 2.0941, 1.8516) & 493 & 3214 \\
 6 & -0.61128 & (3.0000, 3.0000, 3.0000, 2.5084, 2.0331, 1.8219) & 1830 & 8671 \\
 7 & -0.61176 & (3.0000, 3.0000, 3.0000, 2.9744, 2.1658, 1.9910, 1.7925) & 32600 & 107719 \\
\end{array}$$

The computation times reported in Tables 3.2, 3.3, and 3.5 do not include the time spent in the autotuning procedure. In both versions of this problem, and with all the values of $p$ considered, we applied this procedure starting with $\delta = 10^{-5}$ L/h, $h = 1.0$ h, $\kappa = 9$ and $q = 5$ and, using $\sigma = 5$ sample points, arrived at the tuned values of $\delta = 10^{-6}$ L/h, $h = 0.1$ h, $\kappa = 5$ and $q = 3$. The computation times
### TABLE 3.4

RESULTS FOR EXAMPLE 3.1 WITH BOTH SAFETY AND VOLUME CONSTRAINTS AND INITIALIZATION OF $\hat{\phi}$ WITH RANDOM SAMPLING

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\phi^*$ (mol)</th>
<th>$\theta^*$ (L/h) $\times 10^2$</th>
<th>CPU time (s)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-0.54380$</td>
<td>(2.0000)</td>
<td>0.60</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>$-0.58356$</td>
<td>(3.0000, 1.0000)</td>
<td>4.25</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>$-0.59276$</td>
<td>(3.0000, 2.7349, 0.2558)</td>
<td>33.4</td>
<td>284</td>
</tr>
<tr>
<td>4</td>
<td>$-0.59432$</td>
<td>(3.0000, 3.0000, 1.9965, 0.0029)</td>
<td>97.2</td>
<td>701</td>
</tr>
</tbody>
</table>

for autotuning ranged from about 30 s for $p = 1$ to about 950 s for $p = 7$. This suggests that use of this tuning procedure when solving relatively small problems may not be warranted. However, for the type of study considered here, in which optimization is done for increasingly fine control parameterizations, it appears that it is possible to do the tuning for one or two small values of $p$ and then use the results in solving the much more difficult problems with larger $p$. The impact of the tuned parameters on computational performance is considered in more detail in the next two examples. In the 22 problems considered here, $\epsilon^+$ exceeded $\epsilon = 10^{-4}$ mol in five cases, with largest value being $\epsilon^+ = 1.0042 \times 10^{-4}$ mol (for the $p = 3$ case from Table 3.4).
### TABLE 3.5

RESULTS FOR EXAMPLE 3.1 WITH BOTH SAFETY AND VOLUME CONSTRAINTS AND INITIALIZATION OF $\hat{\phi}$ WITH ADJUSTED DOTCVP RESULTS

<table>
<thead>
<tr>
<th>p</th>
<th>$\phi^*$ (mol)</th>
<th>$\theta^*$ (L/h) $\times 10^2$</th>
<th>CPU time (s)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-0.54372$</td>
<td>$(1.9991)$</td>
<td>$0.26$</td>
<td>$3$</td>
</tr>
<tr>
<td>2</td>
<td>$-0.58356$</td>
<td>$(3.0000, 1.0000)$</td>
<td>$2.59$</td>
<td>$24$</td>
</tr>
<tr>
<td>3</td>
<td>$-0.59278$</td>
<td>$(3.0000, 2.7349, 0.2574)$</td>
<td>$24.7$</td>
<td>$218$</td>
</tr>
<tr>
<td>4</td>
<td>$-0.59432$</td>
<td>$(3.0000, 3.0000, 1.9966, 0.0029)$</td>
<td>$49.4$</td>
<td>$369$</td>
</tr>
<tr>
<td>5</td>
<td>$-0.59460$</td>
<td>$(3.0000, 3.0000, 2.8649, 1.1309, 0.0014)$</td>
<td>$532$</td>
<td>$3342$</td>
</tr>
<tr>
<td>6</td>
<td>$-0.59536$</td>
<td>$(3.0000, 3.0000, 3.0000, 2.5084, 0.4835, 0.0046)$</td>
<td>$3320$</td>
<td>$14685$</td>
</tr>
<tr>
<td>7</td>
<td>$-0.59633$</td>
<td>$(3.0000, 3.0000, 3.0000, 2.9530, 2.0437, 0.0029, 0.0004)$</td>
<td>$40093$</td>
<td>$126243$</td>
</tr>
</tbody>
</table>
3.4.2 Example 3.2: Exothermic Series Reaction in a Nonisothermal Semi-batch Reactor

In this section, we consider a temperature optimization problem involving an exothermic series reaction in a nonisothermal semi-batch reactor. The reactor is jacketed and rapid temperature adjustments can be made. However, the cooling capacity is limited, leading to an IPC on the rate of heat generation in the reactor.

Consider the series reaction

\[ A + B \xrightarrow{k_1} C \xrightarrow{k_2} D \]  

(3.10)

occurring in a nonisothermal semi-batch reactor fed a stream containing B, and operated with a temperature profile \( T(t) \). The objective is to maximize the concentration of the product C at the final time \( t_f \) by manipulation of the temperature \( T(t) \). The problem formulation is:

\[
\begin{align*}
\min_{\theta(t)} & \quad \phi = -x_C(t_f) \\
\text{IPCs} & \quad q_{rx}(t) = -\Delta H_1 k_1 x_A(t) x_B(t) V(t) - \Delta H_2 k_2 x_C(t) V(t) \leq q_{rx,\text{max}} \\
\text{s.t.} & \quad \dot{x}_A = -k_1 x_A x_B - \frac{u}{V} x_A \\
& \quad \dot{x}_B = -k_1 x_A x_B + \frac{u}{V} (x_{B,\text{in}} - x_B) \\
& \quad \dot{x}_C = k_1 x_A x_B - k_2 x_C - \frac{u}{V} x_C \\
& \quad \dot{V} = u \\
& \quad k_i = k_{i,0} \exp\left(\frac{-E_i}{RT}\right) = k_{i,0} \exp\left(\frac{-E_i \theta}{R}\right), \quad i = 1, 2 \\
& \quad T \in [293.15, 323.15] \text{ K} \\
& \quad \theta \in [3.095, 3.411] \times 10^{-3} \text{ K}^{-1} \\
& \quad t \in [t_0, t_f] = [0, 0.5] \text{ h}
\end{align*}
\]
Here the state variables are \( x_A(t) \), \( x_B(t) \) and \( x_C(t) \), the concentrations of A, B and C respectively, and the volume \( V(t) \). We find it convenient to use the reciprocal temperature \( \theta(t) = 1/T(t) \) as the manipulated variable. Other model parameters are defined, with values given, in Table 3.6 which also gives the initial states.

In the IPC, \( q_{rx} \) is the rate of heat generation in the reactor. Because of limited cooling capacity, this must not exceed a specified \( q_{rx,\text{max}} \).

The reciprocal temperature \( \theta(t) \) was parameterized with a piecewise constant control profile on an equally spaced mesh with the number of time intervals varying from \( p = 1 \) to \( p = 3 \). The problem was solved to an absolute objective tolerance of \( \epsilon = 10^{-4} \) mol/L. We will use this example to consider the effect on performance of using different values of the IPC resolution parameter \( \omega \) and stepsize \( h \). Since \( \omega \) is used only in the autotuning procedure, in part to determine \( h \), in effect we will study the impact of the stepsize \( h \) on the performance of the global optimization algorithm. For an IPC resolution of \( \omega = 1 \) kJ/h, the tuning procedure gave \( h = 10^{-3} \) h, and for \( \omega = 0.1 \) kJ/h the result was \( h = 10^{-4} \) h. We will also consider the case of \( h = 10^{-5} \) h. Other parameters were autotuned to \( \delta = 10^{-7} \) K\(^{-1}\), \( \kappa = 5 \) and \( q = 3 \). For the case of one time interval \( (p = 1) \), random sampling was used to initialize \( \hat{\phi} \), with the same value used for all step sizes. For \( p > 1 \), we used the adjusted DOTcvp result from the \( p = 1 \) case for the initialization, again with the same value used for all step sizes. We initialized \( \hat{\phi} \) in this way so that these values were not “too good”; this means an increased computational effort, making the effect of using different stepsize values easier to observe.

Results for the \( \epsilon \)-global minimum \( \phi^* \) (rounded to the nearest \( 10^{-5} \) mol/L), for the corresponding \( \epsilon \)-global minimizer \( \theta^* \) (rounded to the nearest \( 10^{-8} \) K\(^{-1}\)), and for \( \epsilon^+ \) are shown in Table 3.7 for the different values of \( h \) considered. We make
## TABLE 3.6

MODEL PARAMETERS FOR EXAMPLE 3.2

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_{1,0}$</td>
<td>Pre-exponential factor</td>
<td>4</td>
<td>L/(mol h)</td>
</tr>
<tr>
<td>$k_{2,0}$</td>
<td>Pre-exponential factor</td>
<td>800</td>
<td>h$^{-1}$</td>
</tr>
<tr>
<td>$E_1$</td>
<td>Activation energy</td>
<td>$6 \times 10^3$</td>
<td>J/mol</td>
</tr>
<tr>
<td>$E_2$</td>
<td>Activation energy</td>
<td>$20 \times 10^3$</td>
<td>J/mol</td>
</tr>
<tr>
<td>$R$</td>
<td>Gas constant</td>
<td>8.3145</td>
<td>J/(mol K)</td>
</tr>
<tr>
<td>$\Delta H_1$</td>
<td>Reaction enthalpy</td>
<td>$-30$</td>
<td>kJ/mol</td>
</tr>
<tr>
<td>$\Delta H_2$</td>
<td>Reaction enthalpy</td>
<td>$-10$</td>
<td>kJ/mol</td>
</tr>
<tr>
<td>$x_{A0}$</td>
<td>Initial concentration of A</td>
<td>10</td>
<td>mol/L</td>
</tr>
<tr>
<td>$x_{B0}$</td>
<td>Initial concentration of B</td>
<td>0.0</td>
<td>mol/L</td>
</tr>
<tr>
<td>$x_{C0}$</td>
<td>Initial concentration of C</td>
<td>0.0</td>
<td>mol/L</td>
</tr>
<tr>
<td>$V_0$</td>
<td>Initial volume</td>
<td>1.0</td>
<td>L</td>
</tr>
<tr>
<td>$x_{B,in}$</td>
<td>Inlet concentration of B</td>
<td>20</td>
<td>mol/L</td>
</tr>
<tr>
<td>$q_{rx,max}$</td>
<td>Maximum heat generation rate</td>
<td>150</td>
<td>kJ/h</td>
</tr>
<tr>
<td>$u$</td>
<td>Feed rate of B</td>
<td>0.35</td>
<td>L/h</td>
</tr>
</tbody>
</table>

The following observations:

1. Comparing the results for $\omega = 1$ kJ/h ($h = 10^{-3}$ h) and $\omega = 0.1$ kJ/h ($h = 10^{-4}$ h), we see that the results for $\phi^*$ differ by more than the tolerance $\epsilon$. The subpaving tolerance $\delta$ is the same in both cases, but with the greater constraint resolution (tighter constraint bounding) in the $\omega = 0.1$ kJ/h ($h = 10^{-4}$ h) case, we observe...
10^{-4} \text{ h}) case, there are many fewer boxes of width less than $\delta$ that remain undecided. That is, more of the search domain can be assigned to $G^{-}$ and removed from $G^{+}$. Thus, in the $h = 10^{-4} \text{ h}$ case, there is a slightly larger $G^{-}$ over which continuous satisfaction of the IPC can be rigorously guaranteed, and so a slightly better value of $\phi^*$ can be found. In using an even smaller $h = 10^{-5} \text{ h}$, no further improvements in $\phi^*$ are noted (differences are within the $\epsilon$ tolerance).

2. For the lowest resolution case of $\omega = 1 \text{ kJ/h}$ ($h = 10^{-3} \text{ h}$), the values of $\epsilon^+$ are significantly larger than $\epsilon$, but, for the higher resolution cases, $\epsilon^+$ is less than or only slightly larger than $\epsilon$. As just explained, higher resolution (smaller $h$) results in a somewhat larger $G^{-}$ and somewhat smaller $G^{+}$. That is, as $h$ is reduced, $G^{-}$ and $G^{+}$ approach each other (both become better approximations of $G$). Thus, as $h$ is reduced, an $\epsilon$-global minimum in $G^{-}$ is more likely to also be an $\epsilon$-global minimum in $G^{+}$.

3. Reducing overestimation of the IPC bounds and objective bounds by reducing the stepsize makes it easier to rigorously eliminate boxes in the IPC test and objective test. This reduces the number of iterations required (number of boxes that must be tested). However, there is a tradeoff—using a smaller stepsize will reduce the number of iterations, but each iteration will require more computation time.

4. When the autotuning option is used, and a specified IPC resolution parameter $\omega$ is used to tune $h$, the use of too loose a resolution (such as in the $\omega = 1 \text{ kJ/h case}$) can increase the computation cost. This may seem counterintuitive, but can be understood in terms of bound overestimation. Toleration
of too much overestimation can have a negative effect on computational performance.
### TABLE 3.7

RESULTS FOR EXAMPLE 3.2

<table>
<thead>
<tr>
<th>ω &amp; h</th>
<th>p</th>
<th>φ* (mol/L)</th>
<th>θ* (K⁻¹) ×10³</th>
<th>ε⁺ (mol/L)</th>
<th>CPU time (s)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ω = 1 kJ/h</td>
<td>1</td>
<td>-1.35686 (3.26923)</td>
<td>9.04×10⁻⁴</td>
<td>61.2</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>h = 10⁻³ h</td>
<td>2</td>
<td>-1.37487 (3.09504, 3.23517)</td>
<td>5.81×10⁻⁴</td>
<td>1330</td>
<td>1687</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-1.38568 (3.09504, 3.09504, 3.21285)</td>
<td>3.20×10⁻⁴</td>
<td>17500</td>
<td>26250</td>
<td></td>
</tr>
<tr>
<td>ω = 0.1 kJ/h</td>
<td>1</td>
<td>-1.35766 (3.26691)</td>
<td>1.051×10⁻⁴</td>
<td>151</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>h = 10⁻⁴ h</td>
<td>2</td>
<td>-1.37536 (3.09504, 3.23329)</td>
<td>&lt; ε</td>
<td>540</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-1.38594 (3.09502, 3.09502, 3.21152)</td>
<td>&lt; ε</td>
<td>1140</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-1.35770 (3.26679)</td>
<td>&lt; ε</td>
<td>1850</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>h = 10⁻⁵ h</td>
<td>2</td>
<td>-1.37541 (3.09502, 3.23311)</td>
<td>&lt; ε</td>
<td>2670</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-1.38596 (3.09504, 3.09504, 3.21145)</td>
<td>&lt; ε</td>
<td>8700</td>
<td>95</td>
<td></td>
</tr>
</tbody>
</table>
3.4.3 Example 3.3: Parallel Reactions in an Isothermal Semi-batch Reactor

In this section, we consider a feed rate optimization problem involving the parallel reactions

\[ A + B \xrightarrow{k_1} C \]
\[ 2B \xrightarrow{k_2} D \]

occurring in an isothermal semi-batch reactor fed a stream containing B, and operating at temperature \( T \). For safety reasons, the concentration of B in the reactor must not be allowed to exceed a specified maximum value. This constraint also helps maintain selectivity to the desired product C. The objective is to maximize the amount of product C at the final time \( t_f \) by manipulating the volumetric flow rate \( \theta(t) \) of the feed stream, which contains B at concentration \( x_{B,\text{in}} \). The problem formulation is as follows:

\[
\begin{align*}
\min_{\theta(t)} & \quad \phi = x_A(t_f)V(t_f) - x_{A_0}V_0 \\
\text{IPC} & \quad x_B(t) \leq x_{B,\text{max}} \\
\text{s.t.} & \quad \dot{x}_A = -k_1x_Ax_B - \frac{\theta}{V}x_A \\
& \quad \dot{x}_B = -k_1x_Ax_B - 2k_2x_B^2 + \frac{\theta}{V}(x_{B,\text{in}} - x_B) \\
& \quad \dot{V} = \theta \\
& \quad \theta \in [0,1000] \mu\text{L/min} = [0,0.001] \text{L/min} \\
& \quad t \in [t_0,t_f] = [0,250] \text{min}
\end{align*}
\]

Here the state variables are \( x_A(t) \) and \( x_B(t) \), the concentrations of A and B, respectively, and the volume \( V(t) \). Other model parameters are defined, and
values given, in Table 3.8, which also gives values of the initial states. The flow rate $\theta(t)$ was parameterized as a piecewise constant profile with $p$ equal time intervals. The problem was solved to an absolute objective tolerance of $\epsilon = 10^{-4}$ mol, using an IPC resolution of $\omega = 10^{-4}$ mol/L. Using steps 1 and 2 of the autotuning procedure, a stepsize of $h = 1.0$ min and a subpaving precision of $\delta = 10^{-7}$ L/min were obtained. We will use this example to study the effect on performance of the tuning parameters $\kappa$ and $q$. Adjusted results from DOTcvp are used to initialize $\hat{\phi}$ in all cases.

### TABLE 3.8

**MODEL PARAMETERS FOR EXAMPLE 3.3**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>Reaction rate constant</td>
<td>0.053</td>
<td>L/(mol min)</td>
</tr>
<tr>
<td>$k_2$</td>
<td>Reaction rate constant</td>
<td>0.128</td>
<td>L/(mol min)</td>
</tr>
<tr>
<td>$x_{A0}$</td>
<td>Initial concentration of A</td>
<td>0.72</td>
<td>mol/L</td>
</tr>
<tr>
<td>$x_{B0}$</td>
<td>Initial concentration of B</td>
<td>0.05</td>
<td>mol/L</td>
</tr>
<tr>
<td>$V_0$</td>
<td>Initial volume</td>
<td>1.0</td>
<td>L</td>
</tr>
<tr>
<td>$x_{B,\text{in}}$</td>
<td>Inlet concentration of B</td>
<td>5</td>
<td>mol/L</td>
</tr>
<tr>
<td>$x_{B,\text{max}}$</td>
<td>Maximum concentration of B</td>
<td>0.06</td>
<td>mol/L</td>
</tr>
</tbody>
</table>
We first fixed the ITS truncation order at $\kappa = 17$ and considered different Taylor model orders $q$, with results for the $\epsilon$-global minimum $\phi^*$ (rounded to the nearest $10^{-5}$ mol/L) and the corresponding $\epsilon$-global minimizer $\theta^*$ (rounded to the nearest $10^{-7}$ L/min) shown in Table 3.9 for $p \leq 4$. We then fixed the Taylor model order at $q = 3$ and considered different ITS truncation orders $\kappa$, with results shown in Table 3.10 again for $p \leq 4$. The ranges of the $\kappa$ and $q$ values used covers values that we have found useful in various other applications [62, 63] involving VSPODE. For each value of $p$, the $\epsilon$-global optimum found is the same, within the $\epsilon$ tolerance, for all the combinations of $\kappa$ and $q$ values.
We make the following observations:

1. Increasing either $q$ or $\kappa$ increases the computational expense. This is consistent with the expected VSPODE performance, as discussed in Section 2.4.3.

2. Changing either $q$ or $\kappa$ has little effect on the number of iterations. In gen-
TABLE 3.10

RESULTS FOR EXAMPLE 3.3 WITH FIXED TAYLOR MODEL ORDER $q = 3$ AND DIFFERENT ITS TRUNCATION ORDERS $\kappa$

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>$p$</th>
<th>$\phi^*$ (mol/L)</th>
<th>$\theta^*$ (L/min) $\times 10^4$</th>
<th>CPU time (s)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>−0.36074</td>
<td>(4.526)</td>
<td>3.46</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>−0.37828</td>
<td>(5.371, 4.420)</td>
<td>15.0</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>−0.38446</td>
<td>(5.749, 4.976, 4.380)</td>
<td>157</td>
<td>981</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>−0.38741</td>
<td>(5.970, 5.306, 4.779, 4.355)</td>
<td>247</td>
<td>1393</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>−0.36074</td>
<td>(4.526)</td>
<td>6.53</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>−0.37834</td>
<td>(5.370, 4.424)</td>
<td>27.6</td>
<td>92</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>−0.38445</td>
<td>(5.748, 4.976, 4.379)</td>
<td>252</td>
<td>845</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>−0.38741</td>
<td>(5.970, 5.301, 4.786, 4.353)</td>
<td>393</td>
<td>1154</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>−0.36074</td>
<td>(4.526)</td>
<td>11.3</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>−0.37835</td>
<td>(5.371, 4.424)</td>
<td>21.5</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>−0.38447</td>
<td>(5.749, 4.976, 4.379)</td>
<td>562</td>
<td>1090</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>−0.38741</td>
<td>(5.970, 5.301, 4.786, 4.355)</td>
<td>894</td>
<td>1488</td>
</tr>
</tbody>
</table>

General, we would expect that use of a relatively high $\kappa$ would permit the use of a larger stepsize in satisfying eq. (2.34) in the first phase of VSPODE. However, in this case, we have already tuned the stepsize to meet a specified IPC tolerance, and thus the stepsize is already sufficiently small that increasing $\kappa$ is not worthwhile. Similarly, by fixing a sufficiently small stepsize, we have effectively enabled use of a small value of $q$.  

71
Finally, using $q = 3$ and $\kappa = 5$, we solved larger versions of this problem, with up to $p = 8$ decision variables. These results are presented in Table 3.11. Again we see that rigorous global optimization for nonlinear problems is in general an NP-hard problem. For the $p = 4$ cases (Tables 3.9 and 3.10) and for $p \geq 6$ (Table 3.11), $\epsilon^+$ exceeded $\epsilon = 10^{-4}$ mol, with largest value being $\epsilon^+ = 1.272 \times 10^{-4}$ mol in the $p = 6$ case.
CHAPTER 4

ROBUST DESIGN AND GLOBAL OPTIMIZATION FOR NONLINEAR DYNAMIC SYSTEMS

4.1 Introduction

In chemical engineering, dynamic processes and control systems often must be designed in a way that is robust to uncertainties in process parameters and to possible disturbances. Two common types of problems are: 1. Obtain a feasible operating region that can satisfy certain safety and/or quality control constraints. 2. Seek the best possible performance in the worst-case scenario in the presence of uncertainties or disturbances.

To address the first type of problems, dynamic processes can be modeled as continuous-time hybrid systems [7] subject to appropriate terminal constraints and path constraints. The design problem considered here is then to rigorously determine robust subregions of the design space; that is, subregions in which all constraints will be satisfied over a finite time horizon, and this will remain true for all possible parameter values and disturbances. Most existing methods [30, 94, 108] for addressing this problem are restricted to linear models. Huang et al. [50] have proposed a region-transition-model (RTM) framework for uncertain nonlinear systems. However, they used difference equations to approximate the evolution of the continuous states, which may not realistically represent the true dynamics of a
nonlinear system. Recently, Lin and Stadtherr [65] extended the RTM framework to continuous-time systems and used it to develop a rigorous approach for safety analysis. In general, their approach was not designed to handle uncertainties, although the approach could be used to address robust design problems by treating uncertain parameters as additional design variables, which introduces unnecessary computational cost.

The second type of problems can often be formulated as min-max dynamic optimization problems. Some min-max optimization problems can be solved using indirect methods. For example, in the context of optimal control, Miele et al. [79, 80] show how to transform min-max problems into the Mayer-Bolza problem of variational calculus. More commonly, some direct numerical approach is applied, perhaps with some useful simplifying assumptions [26]. In general, however, global optimality cannot be guaranteed using those methods. For static min-max problems, Zuhe et al. [125] describe a deterministic $\alpha$-global method based on interval analysis and using branch and bound, and a stochastic global method, namely a genetic algorithm, has been proposed [24] as well. Also, a deterministic global method has recently been described [109] for a closely related static problem. For standard minimization problems with dynamic systems, there have been significant recent advances [61, 62], in deterministic global optimization, and extension of such methods to bi-level optimization problems, such as the min-max problem, has begun [81].

In this chapter, we present a new, rigorous approach for the robust design and optimization of nonlinear, continuous-time dynamic systems. This method can be applied on both types of problems described above and is based on constraint propagation using Taylor-models under a double-layer RTM framework. In the inner
layer, a possible subregion of the design space is tested over different subregions of uncertainty. In the outer layer, the results from the inner layer are collected and summarized to determine overall feasibility, optimality and subsequent action. Within this bi-level framework, we adopt a heuristic sorting procedures. This enables efficient and dynamic reduction of both the uncertainty and decision space domains.

4.2 Problem Statement

The robust design or optimization problem can often be stated in general as follows:

\[
\min_{p \in P} \max_{\theta \in \Theta} \phi(x(t_\mu, \theta, p), \theta, p), \ \mu = 0, 1, \ldots, r
\]

s.t.

\[
\dot{x} = f(x, \theta, p), \ x_0 = x_0(\theta, p), \ t \in [t_0, t_r].
\]

\[
0 \geq g(x, \theta)
\]

\[
0 \geq l(x(t_\mu, \theta, p), \theta, p)
\]

Here \( p \) is the decision parameter vector and \( \theta \) is a vector of uncertain parameters. These are enclosed by given interval bounds \( P \subset \mathbb{R}^{n_p} \) and \( \Theta \subset \mathbb{R}^{n_\theta} \), respectively. For convenience, we will also use the combined vector \( y = (p, \theta)^T \), so \( Y = (P, \Theta)^T \subset \mathbb{R}^{n_p+n_\theta} \). The vector (length \( n_x \)) of state variables in the dynamic system is denoted by \( x \). The objective function \( \phi \) is expressed in terms of the state values at discrete points \( t_\mu, \mu = 0, 1, \ldots, r \), and of the decision and uncertain parameters. If the problem is posed without an objective function, the goal here is to obtain a rigorous inner approximation \( \mathcal{L}_F \) of the entire feasible parameter region \( P_F \) that can satisfy all the path constraints \( g \) and terminal constraints \( l \) and identify the infeasible subregions which violate one or more of these constraints.
on any subspace of $\Theta$. The infeasible subregions are put to $L_I$. Subject to the subpaving tolerance $\delta$, there exists remainder undecided subregions which are stored in $L_U$. The robust design problem without objective function can then be stated as follows:

\[
\begin{align*}
\text{Given : s.t. } & \dot{x} = f(x, \theta, p), \ x_0 = x_0(\theta, p), \ t \in [t_0, t_r]. \\
& 0 \geq g(x, \theta) \\
& 0 \geq l(x(t_{\mu}, \theta, p), \theta, p) \\
& \text{Find :} \\
& P \Rightarrow \begin{cases} \\
L_F &= \{P_1^F, P_2^F, \ldots, P_{n_F}^F\} \\
L_I &= \{P_1^I, P_2^I, \ldots, P_{n_I}^I\} \\
L_U &= \{P_1^U, P_2^U, \ldots, P_{n_U}^U\} \\
\end{cases}
\end{align*}
\]

When the goal is to optimize an objective function for the worst-case scenario, a rigorous optimizer $p^*$ will only be considered inside $L_F$.

The problem formulation [4.1] can also be interpreted as a bi-level problem. Say that, in terms of $p$, the solution to the maximization problem is obtained at $\theta = \theta^i(p)$ with optimal value $\phi = \varphi_{\text{max}}(\theta^i(p), p)$. Then the overall minimization problem can be stated as

\[
\begin{align*}
\min_{p \in \mathcal{L}_F} & \quad \varphi_{\text{max}}(\theta^i(p), p), \ \theta^i(p) \in \Theta \\
\text{s.t.} & \quad \dot{x} = f(x, \theta, p), \ x_0 = x_0(\theta, p), \ t \in [t_0, t_r]. \\
& \quad 0 \geq g(x, \theta) \\
& \quad 0 \geq l(x(t_{\mu}, \theta, p), \theta, p)
\end{align*}
\]

76
In these terms, we seek a rigorous $\epsilon$-global minimum $\phi^* = \varphi_{\text{max}}^*$ at $p = p^*$.

4.3 Solution Procedure

In this section, we present a new deterministic method for the robust design and optimization of continuous-time dynamic systems. The algorithm is constructed on a dynamic branch-and-reduce framework using a constraint propagation procedure (CPP) with Taylor models. Starting from the initial search interval $Y^{(0)} = Y = (P, \Theta)^T$, the algorithm iterates by applying constraint tests to determine the feasibility of each subregions and/or an objective test to identify subregions that may contain points which can improve the incumbent solution $\hat{\varphi}_{\text{max}}$ (upper bound on global minimum of $\varphi_{\text{max}}$). As the subintervals are generated, they are stored in a working list $\mathcal{L}$. The working list is structured following the scheme proposed by Zuhe et al. [125], in which different sublists are created for different parameter subregions. That is, each sublist $S_i$ stores subintervals (boxes) $Y_{i,j} = (P_i, \Theta_{i,j})^T$, $j = 1, \ldots, l$, which share a common decision parameter space $P_i$ but have different uncertain parameter spaces $\Theta_{i,j}$. At each iteration, a tested subinterval is given status flags based on its feasibility test and/or objective test results. Sublists are then flagged and processed (discarded, bisected, sorted and updated) based on the status flag of their member subintervals. The basic components of the algorithm are described below.

4.3.1 Initialization

The first step is to establish the working list $\mathcal{L}$. The working list $\mathcal{L}$ has one initial sublist $S_1^{(0)}$. This sublist initially contains one element, the entire original search space $Y^{(0)} = Y = (P, \Theta)^T$, and is given an initial status flag of
To solve a robust optimization problem, an additional step is to initialize $\hat{\phi}_{\text{max}}$, an upper bound on the global optimum $\phi_{\text{max}}$. The value of $\phi_{\text{max}}$ at any feasible point $\hat{p} \in P$ can be used for this purpose. This point is obtained $\hat{\phi}_{\text{max}}$ simply by choosing some $\hat{p} \in P$ (the choice may be done randomly or based on the user’s knowledge), and then solving the standard maximization problem:

$$
\max_{\theta \in \Theta} \phi(x(t, \theta, \hat{p}), \theta, \hat{p}), \mu = 0, 1, \cdots, r \\
\text{s.t.} \quad \dot{x} = f(x, \theta, \hat{p}), \quad x_0 = x_0(\theta, \hat{p}), \quad t \in [t_0, t_r].
$$

(4.4)

The solution to this problem is then set to be $\hat{\phi}_{\text{max}}$. We solve this problem using the deterministic $\epsilon$-global method described by Lin and Stadtherr [62], specifying the convergence tolerance to be $\epsilon_{\text{in}}$. This “inner tolerance” will also be applied later when updates to $\hat{\phi}_{\text{max}}$ are done. We also set an “outer tolerance” $\epsilon_{\text{out}}$ for use in the objective test (discussed below).

4.3.2 Domain Elimination Rules

Two basic domain elimination rules are used to discard regions of the search space: 1. On a certain decision parameter region $P_i$, if the incumbent $\hat{\phi}_{\text{max}}$ can not be improved by more than the outer tolerance $\epsilon_{\text{out}}$ and/or one or more constraints are violated, then all the subintervals in the corresponding sublist $S_i$ will be discarded. 2. Inside a sublist $S_i$, if it is known that $\Theta_{i,j}$ does not contain the worst-case scenario $\theta^*_i$ for $p \in P_i$ and all the constraints are satisfied, then $Y_{i,j}$ will be removed from $S_i$. 

78
4.3.3 Feasibility Test

In this and next subsection, we consider a subinterval \( Y_{ij}^{(k)} \) (for simplicity, we will subsequently drop the iteration counter superscript \( k \)). By running VSPODE we obtain corresponding interval bounds and Taylor models of constraint functions and then perform path constraint and terminal constraint tests as detailed in Chapter 3. Note that a modified CPP is applied in the robust design and optimization applications. The CPP is carried out on both the decision parameter space \( P_i \) and the uncertain parameter space \( \Theta_{i,j} \). The potential outcomes of the CPP are:

(a) If \( \Theta_{i,j} \) is reduced (partially or entirely) or \( P_i \) is entirely eliminated, this means that for at least part of \( \Theta_i \) the current \( P_i \) does not contain any points at which the constraint can be satisfied and thus \( Y_{i,j} \) is immediately marked False (subinterval flags are not all uppercase).

(b) Otherwise, if \( \Theta_{i,j} \) is not changed, and \( P_i \) is either partially reduced or not changed, then a clear conclusion cannot be drawn. The reduced parameter space \( P_i \) is propagated and used to update all the members in the sublist \( S_i \).

Possible overall outcomes of the feasibility test are:

1. No \( p \in P_{i,j} \) can satisfy one or more of the constraints for all the \( \theta \in \Theta_{i,j} \). \( Y_{i,j} \) is marked False. The False flag will later trigger the elimination of the sublist \( S_i \) that \( Y_{i,j} \) belongs to.

2. Integration terminates at the specified final time \( t_f \) and \( p \in P_{i,j} \) satisfy all the constraints over \( \theta \in \Theta_{i,j} \). \( Y_{i,j} \) is then marked True. It should be emphasized here that the True status does not lead to the conclusion that
\( P_{i,j} \) is a feasible region, because constraints violation may occur over other members subregions in \( S_i \).

3. In one or more constraint tests, a clear (True or False) conclusion cannot be drawn. The rest constraints are satisfied. In this case, \( Y_{i,j} \) is labelled Undecided.

4. Integration terminates early due to failure in the verification phase of VSPODE. In this case, \( Y_{i,j} \) will have a status of Unsolved.

In a robust optimization application, the objective test will then be performed, unless a False flag is signalled or VSPODE integration failed.

4.3.4 Objective Test

After the feasibility test, we evaluate a Taylor model \( T_\phi(y) \) of the objective function and then its interval bound \( \Phi_{i,j} \) for \( y \in Y_{i,j} = (P_i, \Theta_{i,j})^T \). The test then proceeds as follows:

1. If the lower bound of \( \Phi_{i,j} \) is greater than \( \hat{\varphi}_{\text{max}} - \epsilon_{\text{out}} \), then, since the interval bound \( \Phi_{i,j} \) may not even enclose \( \varphi_{\text{max}} \) for \( p \in P_i \), it is known with certainty that \( \varphi_{\text{max}} \) is greater than \( \hat{\varphi}_{\text{max}} - \epsilon_{\text{out}} \) over \( p \in P_i \). In other words, the current decision parameter region \( P_i \) does not contain any points at which \( \hat{\varphi}_{\text{max}} \) can be improved by more than the tolerance \( \epsilon_{\text{out}} \). Thus, the current subinterval \( Y_{i,j} \) is marked as False. The False flag will later trigger the elimination of the sublist \( S_i \) that \( Y_{i,j} \) belongs to. (The feasibility test and the objective test each have their own status flag)

2. If the upper bound of \( \Phi_{i,j} \) is smaller than \( \hat{\varphi}_{\text{max}} - \epsilon_{\text{out}} \), then \( Y_{i,j} \) is marked as True and the objective test is stopped. Note that the True status does not
lead to the conclusion that a better solution lies in $P_1$, because for parameter values $p \in P_1$ the worst-case scenario $\theta_i^\dagger$ may not be encountered over $\Theta_{i,j}$.

3. Otherwise, if $\hat{\varphi}_{\text{max}} - \epsilon_{\text{out}} \in \Phi_{i,j}$, we use $T_\phi(y)$ in a modified CPP (detailed in 4.3.3) based on the constraint $\phi(y) - \hat{\varphi}_{\text{max}} + \epsilon_{\text{out}} \leq 0$. This may result either a False flag or an Undecided flag.

4.3.5 Sublist Management

In this step, we consider a sublist $S_i$ corresponding to the decision parameter region $P_i$. The status of the sublist depends on the feasibility and objective test results of its members $Y_{i,j}$, $j = 1, \ldots, l$. Based on the results, we proceed as follows (Steps used in only design or only optimization problems are labeled respectively with “a” and “b”):

1. If any $Y_{i,j}$ is marked False either in feasibility test or objective test, then a better solution does not exist in $P_1$ or $P_1$ cannot be feasible and thus $S_i$ is marked FALSE. There is no need to further investigate the remaining items in $S_i$.

2. If any $Y_{i,j}$ is flagged Unsolved, this indicates that the current subinterval is too large for VSPODE. $S_i$ remains UNSOLVED. This case is likely to happen only in the early stage of algorithm execution.

3. If all the members $Y_{i,j}$, $j = 1, \ldots, l$, have True flags in feasibility test and objective test (if any). Mark $S_i$ as TRUE.

4. Otherwise, some of the subinterval flags are Undecided and the others are True or all the flags are Undecided. In these cases, mark $S_i$ as UNDECIDED.
Note that if all the members in $S_i$ passed feasibility test, it will only be submitted to objective test in later iterations.

5a In robust design problems without an objective function, if $S_i$ is marked FALSE, put $P_i$ into infeasible region list $L_I$ and remove $S_i$. If $S_i$ is marked TRUE, put $P_i$ into feasible region list $L_F$ and remove $S_i$. Otherwise, submit $S_i$ for bisection.

5b In robust optimization applications, if $S_i$ is marked TRUE, then any point in $P_i$ can be used to replace the incumbent $\hat{p}$ and obtain a new value of $\hat{\varphi}_{\text{max}}$ that improves the incumbent value by more than the tolerance $\epsilon_{\text{out}}$. If $S_i$ is marked FALSE, it will be removed. Otherwise, submit $S_i$ for bisection.

6b The objective function bounds $\Phi_{i,j}$ are known for all members of an UN-DECIDED or TRUE sublist. The second domain elimination rule can be applied inside such an $S_i$ by analyzing these bounds. First, find the subinterval $Y_{i,\alpha}$ whose lower objective function bound is the greatest; that is $\alpha = \arg \max_j \{ \Phi_{i,j} \}, j = 1, \ldots, l$. Then, remove any feasible subintervals $Y_{i,\beta}$ from $S_i$, if $\Phi_{i,\beta} < \Phi_{i,\alpha}$, since it is not possible for the worst-case scenario $\theta^*$ to be in $\Theta_{i,\beta}$. After clearing the discarded subintervals, we rearrange the remaining subintervals in order of decreasing $\Phi$ value so that later tests will start from the boxes with greater $\Phi$, and thus be more likely to yield a False flag without need to test the entire sublist.

7b Finally, we obtain interval bounds $\Phi_i$ on the objective function over the new sublist $S_i$ by taking the interval hull of $\Phi_{i,j}$ over all subintervals in $S_i$. Then, we rearrange the all sublists $S_i, i = 1, \ldots, m$, in a such way that UNSOLVED sublists will be put at the front (top) of $L$ and that the remaining sublists
will be sorted in order of increasing $\Phi$. Thus, sublists that appear more likely to contain a better solution are encountered sooner.

4.3.6 Update Incumbent Solution and Sublists

To update $\hat{\varphi}_{\text{max}}$ using a TRUE sublist $S_i$, the midpoint $m(P_i)$ is chosen as $\hat{p}$ and a new $\hat{\varphi}_{\text{max}}$ is computed by solving the standard maximization problem stated in eq. (4.4) over the uncertain parameter subspaces $\Theta_{i,j}$, $j = 1, \ldots, l$. Since the interval bounds on the objective function are known for all elements of TRUE and UNDECIDED sublists, the new $\hat{\varphi}_{\text{max}}$ is then used immediately to update these sublists. If $\hat{\varphi}_{\text{max}} - \Phi_{i,j} \leq \epsilon_{\text{out}}$ over $Y_{i,j}$, then mark the corresponding sublist $S_i$ as FALSE. For a sublist $S_i$ previously labeled TRUE, if $\hat{\varphi}_{\text{max}} - \epsilon_{\text{out}} \in \Phi_{i,j}$, over one subinterval $Y_{i,j}$, then mark $S_i$ as UNDECIDED.

4.3.7 Bisection

To keep the bi-level structure of $\mathcal{L}$, bisection decisions are made sublist-wise; that is, all the boxes in the same sublist will be bisected at once. When bisecting on an uncertain parameter, it is done within a particular sublist. When bisecting on a decision parameter, a new sublist is created for one of the resulting parameter subregions and the other one stays in the original sublist. Other bisection rules are heuristics set for performance. If the decision parameter space has just been sufficiently reduced by a CPP (more than 50% by volume), then return the sublist without bisection. Select the component with the largest relative width for bisection. After bisection, we also reinitialize status flags for the sublist (as UNSOLVED) and for its subintervals (as Unsolved). Subject to subpaving tolerance $\delta$, tiny UNDECIDED sublist will be removed from $\mathcal{L}$ and the corresponding
parameter region will be stored in undecided list $L_U$.

4.3.8 Overall Optimization Algorithm

A simplified version of algorithm can be stated as follows:

1. Initialize (Section 4.3.1).

2. Remove from the front of $L$ the sublist $S_1$ for testing. This sublist may have status TRUE, UNDECIDED, FALSE or UNSOLVED.
   (a) If TRUE, go to step 4.
   (b) If UNDECIDED, go to step 5.
   (c) If FALSE, go to step 6.
   (d) Otherwise, proceed to step 3.

3. Send $Y_{1,j}$, $j = 1, \ldots, l$ from $S_1$ to the feasibility test (Section 4.3.3) and objective test (Section 4.3.4) and process $S_1$ (Section 4.3.5). This sublist may now have status of TRUE, UNDECIDED, FALSE or UNSOLVED.
   (a) If TRUE or UNDECIDED, go to step 7.
   (b) If FALSE, go to step 6.
   (c) Otherwise, go to step 5.

4. Update $\hat{\phi}_{\text{max}}$ and the sublists (Section 4.3.6); go to step 7.

5. Bisect $S_1$ (Section 4.3.7); go to step 7.

6. Discard $S_1$, go to step 7.

7. If $L$ is empty, proceed to step 8. Otherwise, return to step 2.

8. Terminate.
   (a) In robust design applications, the initial design parameter space $P^0$ is categorized into feasible subregion list $L_F$, infeasible subregion list $L_I$. 84
and undecided subregion list $L_U$. (b) In robust optimization applications, if $L_U = \emptyset$ the final $\hat{p}$ represents an $\epsilon_{\text{out}}$-global minimum with respect to the decision parameters $p$ of $\varphi_{\text{max}}$, which is an $\epsilon_{\text{in}}$-global maximum of $\phi$ with respect to the uncertain parameters $\theta$. Otherwise, an optional tuning procedure (detailed in Chapter 3) can be applied to further investigate the undecided region.

4.4 Computational Studies

In this section, four example problems are presented to discuss different computational aspects. All examples were solved on a single core of Intel i7 2.2GHz machine. VSPODE with a $k = 17$ order interval Taylor QR method and a $q = 5$ order Taylor model, was used to integrate the dynamics system.

4.4.1 Example 4.1: Parameter Estimation

Consider a robust parameter estimation problem for a Lotka-Volterra model, formulated as follows:

$$\min_{\hat{p}} \max_{\theta} \phi = \sum_{\mu=1}^{10} (x_2(t_{\mu}, \theta, \hat{p}) - \bar{x}_{2,\mu})^2$$

$$\dot{x}_1 = p_1 x_1 (a - x_2)$$

$$\dot{x}_2 = p_2 x_2 (x_1 - b)$$

$$x_0 = (\theta, x_{2,0})^T$$

$$t \in [t_0, t_{10}] .$$

This ODE model is a standard representation of predator-prey interactions in
TABLE 4.1

MODEL PARAMETERS FOR EXAMPLE 4.1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>[2.9, 4.0]</td>
</tr>
<tr>
<td>$p_2$</td>
<td>[0.9, 3.0]</td>
</tr>
<tr>
<td>$\theta$</td>
<td>[1.0, 1.4]</td>
</tr>
<tr>
<td>$a$</td>
<td>1.0</td>
</tr>
<tr>
<td>$b$</td>
<td>1.0</td>
</tr>
<tr>
<td>$x_{2,0}$</td>
<td>1.1</td>
</tr>
</tbody>
</table>

ecology, with $x_1$ representing prey population and $x_2$ predator population. In this case, a precise measurement of the prey population $x_1$ is not available, and only a range of the initial population $x_{1,0} = \theta \in [1.0, 1.4]$ can be considered known. Therefore, the estimation of the parameters $p_1$ and $p_2$ is based on measurements $\tilde{x}_{2,\mu}$ at times $t = t_\mu$, $\mu = 1, 2, \ldots, 10$, of the predator population. This measurement data was generated \[33\] from the model using a fixed value $x_{1,0} = 1.2$ and $p = (3, 1)^T$, with a small amount of normally distributed random error (zero mean and standard deviation 0.01) added. Instead of treating $x_{1,0}$ as an extra parameter to fit and solving a standard parameter estimation problem, we consider $x_{1,0}$ as an uncertain variable and aim to find the best-fitting parameters $p_1$ and $p_2$ in the worst-case scenario with respect to the uncertainty in $\theta = x_{1,0}$. The model parameters are listed in Table \[4.1\]. Using tolerances $\epsilon_{in} = \epsilon_{out} = 0.00001$ and starting with $\hat{p} = (2.95, 2.0)^T$, the approach described above converged to a
solution of $\phi^* = \varphi_{\text{max}}^* = 0.04874$ with the parameter values $p^* = (3.2815, 0.9000)^T$
and worst-case uncertain parameter value $\theta^\dagger = 1.0000$. This required about 760 CPU seconds and 9789 calls of VSPODE.

4.4.2 Example 4.2: Batch Reactor Design

In this example, we consider a first-order exothermic reaction in a batch reactor fitted with a cooling jacket. The goal is to find feasible heat transfer coefficient range that can avoid overheat and lead to desired conversion over an uncertain initial temperature range. The model is given by the following mass and energy balances:

\[
\begin{align*}
\dot{X} &= k_0 \exp \left( -\frac{E_a}{RT} \right) (1 - X) \\
\dot{T} &= \frac{UA}{C_{A_0} V C_p} (T_a - T) - \frac{\Delta H_R k_0}{C_p} \exp \left( -\frac{E_a}{RT} \right) (1 - X) \\
T &\leq T_{\text{over}} \\
X_t &> X_d
\end{align*}
\] (4.6)

The model parameters are listed in Table 4.2. We solve this problem first using the robust design algorithm outlined in Section 4.3.8 and then by the “overall scan” approach [65]. The subpaving tolerances in both runs are set $\delta = 0.5$ for $T_0$ and $\delta = 0.01$ for $UA$. The results are presented in Table 4.3. The difference between feasible ranges obtained from the two approaches is less than the specified subpaving tolerance $\epsilon = 0.01$. The robust design approach costs significantly less CPU time and VSPODE calls than the overall scan approach, since the bi-level robust design algorithm is implemented with more elaborate domain reduction and sublist management procedures to avoid redundant work. This is illustrated
TABLE 4.2

MODEL PARAMETERS FOR EXAMPLE 4.2

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
<td>Initial temperature</td>
<td>[460, 510] K</td>
</tr>
<tr>
<td>$UA$</td>
<td>Heat transfer constant</td>
<td>[2, 4] W/K</td>
</tr>
<tr>
<td>$E_a$</td>
<td>Activation energy</td>
<td>6000 J/mol</td>
</tr>
<tr>
<td>$k_0$</td>
<td>Kinetic rate constant</td>
<td>0.022 s(^{-1})</td>
</tr>
<tr>
<td>$C_{A0}$</td>
<td>Initial concentration of A</td>
<td>10 mol/m(^3)</td>
</tr>
<tr>
<td>$V$</td>
<td>Volume of the reactor</td>
<td>0.1 m(^3)</td>
</tr>
<tr>
<td>$C_p$</td>
<td>Total heat capacity</td>
<td>60 J/mol K</td>
</tr>
<tr>
<td>$R$</td>
<td>Gas constant</td>
<td>8.314 J/mol/K</td>
</tr>
<tr>
<td>$\Delta H_R$</td>
<td>Heat of reaction</td>
<td>-140,000 J/mol</td>
</tr>
<tr>
<td>$T_a$</td>
<td>Coolant temperature</td>
<td>298 K</td>
</tr>
<tr>
<td>$t_f$</td>
<td>Reaction time</td>
<td>1500 s</td>
</tr>
<tr>
<td>$X_d$</td>
<td>Desired Conversion</td>
<td>97.5%</td>
</tr>
<tr>
<td>$T_{over}$</td>
<td>Overheat temperature</td>
<td>540 K</td>
</tr>
</tbody>
</table>

in more detail in Figure 4.1. Feasible, infeasible and undecided regions are represented with color green, red and black respectively. As shown in Figure 4.1b, the blank regions were not investigated, since the conclusions on the corresponding design parameter $UA$ range have already been drawn based on the investigation of other regions.
Figure 4.1. Parameter region subpaving plot of the batch reactor design example

TABLE 4.3

RESULTS FOR EXAMPLE 4.2

<table>
<thead>
<tr>
<th></th>
<th>Robust Design</th>
<th>Overall Scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasible $UA$ region</td>
<td>[2.889, 3.110]</td>
<td>[2.889, 3.111]</td>
</tr>
<tr>
<td>% of undecided region</td>
<td>1.7%</td>
<td>1.7%</td>
</tr>
<tr>
<td>CPU seconds</td>
<td>60</td>
<td>750</td>
</tr>
<tr>
<td>VSPODE calls</td>
<td>153</td>
<td>1923</td>
</tr>
</tbody>
</table>
4.4.3 Example 4.3: Saddle Point

For an unconstrained robust optimization, the global solution is always be a saddle point. In this example, we considered an unconstrained robust optimization of a linear ODE system with an available analytical solution. The problem is formulated as follows:

$$\min_p \max_{\theta} \phi = x_1(a - \theta) - x_2(b - p)$$

$$\dot{x}_1 = x_1 + \theta x_2$$

$$\dot{x}_2 = px_1 + x_2$$

$$x_0 = (x_{1,0}, x_{2,0})^T$$

$$t \in [t_0, t_f].$$

(4.7)

The model parameters are listed in Table 4.4. Using tolerances $\epsilon_{in} = \epsilon_{out} = 0.0001$ and starting with $\hat{p} = 0.5$, the approach described above converged to a solution of $\phi^* = \varphi_{max}^* = 3.31e-6$ with the parameter values $p^* = 0.116809$ and worst-case uncertain parameter value $\theta^* = 0.11609$. This required 21 CPU seconds and 2512 calls of VSPODE. The objective value surface is obtained from analytical solution of the ODE system and plotted in Figure 4.2. As marked by the orange point in the figure, the solution is a saddle point.
TABLE 4.4

MODEL PARAMETERS FOR EXAMPLE 4.3

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta$</td>
<td>[-1.0, 1.0]</td>
</tr>
<tr>
<td>$p$</td>
<td>[-1.0, 1.0]</td>
</tr>
<tr>
<td>$a$</td>
<td>1.0</td>
</tr>
<tr>
<td>$b$</td>
<td>1.0</td>
</tr>
<tr>
<td>$x_0$</td>
<td>{1.0 1.0}</td>
</tr>
<tr>
<td>$t_0$</td>
<td>0.0</td>
</tr>
<tr>
<td>$t_f$</td>
<td>2.0</td>
</tr>
</tbody>
</table>
4.4.4 Example 4.4: Haldane Law Bioreactor

In this example, we consider a Haldane kinetics nonlinear bioreator model. The goal is to find optimal dilution rate to maximize the worst-case biomass concentration over an uncertain inlet substrate concentration range. The microbial growth equations are:

\[
\begin{align*}
\min_D \max_{S_i} \phi &= -X(t_f) \\
S &> S_{\text{low}} \\
\dot{X} &= (\mu - \alpha D)X \\
\dot{S} &= D(S_i - S) - k\mu X \\
\mu &= \frac{\mu_m S}{K_S + S + K_1S^2}
\end{align*}
\]
TABLE 4.5

MODEL PARAMETERS FOR EXAMPLE 4.4

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Biomass resistance to washout</td>
<td>0.5</td>
</tr>
<tr>
<td>$\mu_m$</td>
<td>Maximum growth rate</td>
<td>1.21 day$^{-1}$</td>
</tr>
<tr>
<td>$k$</td>
<td>Yield coefficient</td>
<td>10.53 g substrate/g cells</td>
</tr>
<tr>
<td>$K_S$</td>
<td>Saturation constant</td>
<td>7.09 g/L</td>
</tr>
<tr>
<td>$D$</td>
<td>Dilution rate</td>
<td>[0.0, 0.2] day$^{-1}$</td>
</tr>
<tr>
<td>$K_I$</td>
<td>Inhibition parameter</td>
<td>1.5 L/g</td>
</tr>
<tr>
<td>$S^i$</td>
<td>Feed concentration of substrate</td>
<td>[4.9, 5.7] g/L</td>
</tr>
<tr>
<td>$X_0$</td>
<td>Initial concentration of cells</td>
<td>0.83 g/L</td>
</tr>
<tr>
<td>$S_0$</td>
<td>Initial concentration of substrate</td>
<td>0.80 g/L</td>
</tr>
<tr>
<td>$t_f$</td>
<td>Total reaction time</td>
<td>5 day</td>
</tr>
<tr>
<td>$S_{low}$</td>
<td>Lower limit on substrate concentration</td>
<td>0.5 g/L</td>
</tr>
</tbody>
</table>

The model parameters are listed in Table 4.5. Two tests were performed on this model. In the first test, we solved the problem using a set of different objective tolerance value. The results are presented in Table 4.6. When use a tight tolerance, a better solution is obtained. This requires investigation at a higher resolution on the parameter space and thus more VSPODE iterations. In the second test, we fixed the objective tolerance at $\epsilon = 1.0e^{-3}$ and considered relaxed path constraint.
### TABLE 4.6

RESULTS FOR EXAMPLE 4.4 USING DIFFERENT OBJECTIVE TOLERANCE $\epsilon$

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$\phi^*$</th>
<th>Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0e-2</td>
<td>-0.84409</td>
<td>169</td>
</tr>
<tr>
<td>1.0e-3</td>
<td>-0.84935</td>
<td>820</td>
</tr>
<tr>
<td>1.0e-5</td>
<td>-0.84984</td>
<td>1832</td>
</tr>
</tbody>
</table>

with $S_{low} = 0.0$ and $S_{low} = 0.25$. The results are presented in Table 4.7. A lower objective value is obtained, when relax the path constraint. This indicates that the constraint is active over the parameter range.
**TABLE 4.7**

RESULTS FOR EXAMPLE 4.4 USING DIFFERENT PATH

CONSTRAINT $S_{\text{low}}$

<table>
<thead>
<tr>
<th>$S_{\text{low}}$</th>
<th>$\phi^*$</th>
<th>Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>-0.90542</td>
<td>1126</td>
</tr>
<tr>
<td>0.25</td>
<td>-0.88097</td>
<td>1457</td>
</tr>
<tr>
<td>0.5</td>
<td>-0.84935</td>
<td>820</td>
</tr>
</tbody>
</table>
CHAPTER 5

DYNAMIC LOAD BALANCING

5.1 Introduction

Rigorous Taylor models method provides guaranteed results of root-finding and global optimization problems for dynamic systems. However, this advantage may come at a high computational cost, since the iterative bounding procedure must be applied over the complete searching space and the number of arithmetic operation calls depends on the order of interval Taylor series and the size of Taylor model (which can both be large). Process engineering problems can have multiple decision variables over a considerable searching range and meanwhile require an immediate solution. A single core or CPU may not perform fast enough. Fortunately, high performance computing (HPC) infrastructures and algorithms are developed to speed up computation. In this dissertation, we present how to apply parallel computation framework on the top of rigorous Taylor models algorithms.

5.2 Background

5.2.1 Parallel Computation

Traditionally, computer programs have been coded for serial computation in which an algorithm is implemented as a serial stream of instructions. These instructions are executed one at a time on a single central processing unit (CPU)
on one computer. On the other hand, parallel computation breaks a problem into independent subproblems or compartments and utilizes multiple processing elements simultaneously to address them. The processing elements can be uniform or diverse and include resources such as a single multi-core processor, a single computer with multiple processors, several networked cluster or grids computers, specialized hardware, or a certain combination of them. It is worth pointing out that due to increasing power consumption, increasing single CPU performance by frequency ramping was no longer the dominant force in processor manufacturing. The industry has been shifted to achieving performance gains with enabling parallel computation on multi-core processors. Multi-core processors are commonly equipped in personal computers and cluster servers.

5.2.2 Branch and Reduce Framework

Domain reduction techniques such as interval Newton method and Taylor model constraint propagation are often applied on a branch and bound framework. Such a rigorous Taylor model algorithm can thus be considered as a type of branch and reduce (BR) approach. In the problems investigated using the BR framework, the tasks on different branches of the search tree are often independent of each other in most iterations (except occasional incumbent solution update in optimization applications).

5.2.3 Parallelism Identification

In the implementation of rigorous Taylor model methods, several options are available to introduce parallel computation using established methods. On a fine-grain level, basic interval arithmetic operations \cite{97} and/or polynomial operations
can be parallelized. On the coarse-grain level, the subintervals generated by BR procedure can be investigated in parallel [8, 42, 102, 103, 124]. Fine-grain parallel implementations divide an entire task into many short duration tasks, while the coarse-grain procedure has significantly fewer tasks, but each task takes longer time to complete. Finer granularity increases the total amount of work that can be simultaneously carried on and thus may lead to faster completion. However, it requires more time for communication and synchronization between nodes (processes or threads) and may potentially be slower. In this application, we choose only to parallelize BR procedure considering that the total communication work required to transmit intermediate interval or Taylor polynomial results will far outweigh that required to pass only the decision subintervals.

Although BR framework is amenable to parallel processing, significant speedup will be achieved only if the computation workloads are evenly distributed to each nodes. However, in general, such a load balance cannot be obtained beforehand by simply dividing initial searching domain into equal volume subintervals and assign them to each node, since as the BR proceeds, the subinterval of interest (containing optimum or root) might be further divided into many children subinterval for tests and the corresponding node may thus become overloaded, while the one that encloses no solutions will be dropped immediately and the corresponding node may then stay idle. Therefore, it is desirable to have a mechanism which can dynamically allocate subintervals over the nodes through BR execution. This type of mechanism is often referred to as dynamic load balancing (DLB).
5.2.4 Multi-process DLB

Most DLB strategies are implemented based on communication between multiple processes. In computing, a process is an instance of a computer program that is being executed. It contains the program code and its current activity [114]. Message passing models and tools are developed for processes to send and receive messages (comprising bytes, data structures, or even segments of code) to other processes. Early process based DLB works \[40, 103, 106, 118\] follow the “manager-worker” scheme in which a central “manager” process distributes workloads to “worker” processes, and controls global termination by sending end signal to the “workers” when the workload pool becomes empty. This scheme is relatively easy to implement and thus has been widely used. However, when applied on a large number of processes, the centralized scheme may not scale up well, since the “manager” process becomes a communication bottleneck. To address the bottleneck issue, more recent decentralized schemes are developed \[42, 69\]. These schemes adopt a distributed strategy that allows each processes to have local workload traffic with neighbour processes instead of “employing a manager” for global work load pooling. In this dissertation, two distributed algorithms will be applied on top of the BR procedure of global optimization algorithm.

5.2.5 Multi-thread DLB

Another type of multiple threads (multi-thread) based DLB implementation on modern multiprocessor PC has recently started to attract more research attention \[58, 124\]. A thread is a sequence of instructions that can be managed independently by an operating system. A collection of interactive threads can be bundled in a single process \[121\]. Although the implementation of threads
and processes differs from one system to another, in general, the use of threads on shared memory multiprocessor is expected to have two advantages over processes: 1. Multiple threads within one process run in the same address space, and thus can share common variables instead of making their own copies. 2. Inter-thread communication on shared memory is more efficient compared with inter-process communication, since no intermediate memory copy is required [5]. Therefore, threads based DLB strategies usually have workload stack(s) stored in shared memory (though possible to use thread local storage). The common workload stack(s) and other shared resources must be protected from concurrent modifications in different threads of execution. Mutual exclusive (Mutex) access algorithms [27, 60] and devices [51, 83] can be implemented to guard shared resource. The performance and scalability of thread based DLB schemes have been tested using a variety of benchmarks [3, 20, 55, 78] and finite element problems [45]. Such tests are performed either on a shared memory system [3, 78] or on a multi-threads supported architecture [110] over a distributed memory environment [20, 45, 55]. In the context of interval related research, the scalability of thread based DLB schemes have not been fully investigated previously, probably due to the machine limitation, while efforts have already been made to improve the performance on a single multiprocessor PC through adjusting nodes virtual network [124] or optimize workload list structure [58]. In this dissertation, a multi-thread based DLB scheme is presented, tested and compared with process based DLB scheme on a single multi-core cluster server node.

Although the underlying implementations of thread based DLB and process based DLB are different, to some degree, it is convenient here to describe them using a common workflow consisting of five steps: workload initialization, virtual
network construction, working statues exchange, workload placement, and global termination.

5.3 General DLB Workflow

5.3.1 Workload Initialization

In the initialization step, a workload measurement index needs to be decided and will be used as a criterion throughout the execution of a DLB scheme. Since such an index is evaluated frequently to determine workload traffic, it must be inexpensive to obtain, and meanwhile lead to easy workload placement decisions. In the interval BR applications, a simply yet effective approach is to use the stack size (number of subintervals) as the work index. This index is chosen here for two reasons: 1. Stack size indicates the current (near future) work loading level on a node, and an empty stack indicates that the node is in/approaching an idle state; 2. A more sophisticated workload measurement may not be necessary, since the goal of DLB is to merely keep nodes occupied with work rather than maintain a real-time even distribution of loads. Another task in the step is to initialize workloads on each node. This is done in different ways depending on the purpose of tests and will be specified later.

5.3.2 Virtual Network Construction

An additional step of initialization is to build the virtual network “infrastructure” for workload distribution and working status exchange. In this step, each node will first have its cooperating nodes (workers/manager or neighbors) specified. The cooperating nodes are a set of nodes directly participating in work status communication and work load transmission with a local node. Correspond-
Figure 5.1. All-to-all virtual network

Status flags and workload buffers are then initialized. Four virtual network configurations considered here are all-to-all, worker/manager, 1-D ring and 2-D Mesh and are illustrated in Figure 5.2-5.4. The network structure used can have a significant effect on network communication overhead. There is a trade-off between message diffusion distance and communication cost in choosing a network structure. On one hand, a structure with more connections per node has a smaller message diffusion distance. This facilitates the status information propagation and the load balance among nodes. On the other hand, increasing the number of connections means more communication required between nodes. This may result in more cost on message transmission and thus degrade the overall performance of DLB strategy. In practice, the optimal network structure depends on factors such as the hardware environment, message transmitting scheme, and the computational cost per load.
Figure 5.2. Worker manager virtual network

Figure 5.3. 1-D ring virtual network
5.3.3 Working Statues Exchange

Decisions on workload placement and global termination are made according to working status of nodes. Each node thus needs to inform cooperating nodes of its local work index through virtual network and record updated work indexes of other nodes in a vector. A common global communication scheme can be applied on all-to-all virtual network in which all the other nodes are included as cooperating nodes. This could provide a real-time global work index vector at the synchronization point, but might come at the expense of intensive message traffic and blocking communication overhead. An alternative scheme is to include only a small subset of the available nodes as cooperating nodes. For example, in the 1-D ring and 2-D mesh network, a node will have direct information exchange with only the nearest neighbors using cheap local point-to-point communication operations. However, in this case, if asynchronous communication is applied, more elaborate algorithms are usually required to keep load balance and determine global termination.
5.3.4 Workload Placement

After obtaining updated work index(es) from cooperating nodes, DLB algorithm decides whether a workload placement is needed to keep workload balance. Such decisions can be made periodically (e.g., after a number of iterations proceed or a certain amount of time elapse) or triggered by events. In the latter case, depending on the event, the decision procedures can be categorized into three types: a sender-initiate approach [112, 120], a receiver-initiate approach [38, 119], and a symmetric approach [22, 68]. In the sender-initiate approach, when the working stack size of a node exceeds an upper limit, this overloaded node will ask other cooperative nodes to share its burden. The receiver-initiate scheme operates in the opposite direction by having an underloaded node request work from loaded nodes, when the work stack of this node is (approaching) empty. The symmetric scheme allows simultaneous and independent requests from both overloaded and underloaded node and thus can be considered as a combination of the previous two approaches.

If a workload placement is necessary, the donor node then decides how much work should be transferred and where to distribute workloads. It is desirable that the amount of offloading work is neither too little nor too much, so that after load balancing both the sender and the receiver can possibly stay busy for at least a few iterations without further load balancing operations. Several workload adjusting rules are available [25, 32, 41, 122]. One simple way is to send a fixed number of loads upon initiating or receiving a request (e.g. work stealing strategy [11]). More sophisticated diffusion rules [25, 46] consider the work indexes on both sides and dynamically adjust the quantity of outgoing loads using a scheme analogous to physical diffusion. Alternatively, the work placement procedure can
be simplified by offloading to a randomly chosen neighbor node. A random scheme may thus be favorable when each node has a relatively large set of cooperating nodes. An additional step is to select loads for transmission. Many heuristic rules have been proposed to select appropriate loads, which could perhaps supply adequate work to the receiver. The selection rule applied can have a significant impact on a parallel BR algorithm for the global optimization problems, since the rule affects evaluation order of workload units and in turn affects the time at which a good objective bound can be found. Holding a good objective bound, it is more likely that subintervals can be discarded using only an objective range test without further bisection and investigation. Therefore, in general, the earlier a good objective bound can be identified, the earlier the global optimization can be completed. However, the exact effect of a selection rule may not be easily foreseen before the execution of a DLB scheme.

5.3.5 Global Termination

The last step of DLB scheme is to detect global termination. For a synchronous parallel algorithm, this can be easily done through periodic blocking communication. However, this is a more complex task for an asynchronous distributed algorithm, since without central control or global communication, it is difficult to confirm that there is no workload left on the nodes or in the communication channels. Dijkstra’s token algorithm is commonly used to provide reliable termination decision \[28, 29, 59\].
5.4 Implementation with Message Passing Interface (MPI)

MPI is a standardized and portable message-passing system developed to operate on a wide variety of parallel computation platforms. This system provides essential virtual network, synchronization, and communication functionality between processes (that have been mapped to processors) in a language-independent way, with language-specific bindings and features. For maximum performance, each CPU (or core in a multi-core machine) will usually be assigned only a single process. This assignment occurs at runtime through the agent that starts the MPI program, normally called mpirun or mpiexec. Several MPI implementations are available. Here the popular implementations MPICH2 is applied. This version includes the following communication functions and objects:

**Communicator** objects connect groups of processes in the MPI session. Each communicator gives each contained process an independent identifier and arranges its contained processes in an ordered topology. MPI understands single group intracommunicator operations as well as bilateral intercommunicator communication. In this context, all the processes are contained in one communicator.

**Allgather** function collects data from all processes and distribute the combined data (vector) to all processes. This is a synchronous blocking all-to-all communication function.

**Send** function performs a blocking send which blocks the execution of source process until the message is received by the destination process.

**Recv** function performs a blocking receive which blocks the execution of destination process until the message is received from the source process.
**Request** objects are used to identify nonblocking communication operations and match the operation that initiates the communication with the operation that terminates it. A request object identifies various properties of a communication operation, such as the send mode, the communication buffer that is associated with it, its context, the tag and destination or source arguments. In addition, this object stores the status of the pending communication operation.

**Isend** function initiates a nonblocking send. The initiating process will proceed after this operation.

**Irecv** function begins a nonblocking receive. The initiating process will proceed after this operation.

**Wait** function blocks a process until a non-blocking send or receive operation has completed.

**Test** function returns whether a send or receive request has been completed.

In this section, we describe two DLB algorithms (SWS and ADLB) constructed using these MPI functions.

5.4.1 Synchronous Work Stealing (SWS)

SWS algorithm applies a global strategy on an all-to-all virtual network. In this scheme, computation and communication are interleaved. Allgather function is called periodically (after a certain time or some number of iterations) to provide synchronization for all the processes and update the global work index vector. At the communication phase, each idle processor (if exist) steals one unit of workload (subinterval) from the most overloaded processes using a receiver-initiate scheme.
This step is implemented using Send andRecv operations. The SWS algorithm is illustrated in Figure 5.5.

Although the global Allgather function in this approach provides a clear global view of workload distribution and thus leads to easy decisions on workload placement and global termination, this comes with a potential drawback of increased communication costs when using many processes. Additional communication overhead may arise due to the synchronous and blocking properties of the scheme. If a process has reached a synchronization point, it then have to wait for other processes to reach the same point and thus actually stay idle during the waiting period. When blocking operation is applied, the sender process will proceed only after confirmed that the data has been transmitted to the receiving buffer. These issues together can degrade overall performance and result in poor scalability of SWS algorithm.
After I iterations
Exchange working
status information
Determine workload
placement
Workload transfer
Resume tests

Figure 5.5. Illustration of SWS scheme

5.4.2 Asynchronous Diffusive Load Balancing (ADLB)

The ADLB scheme follows a local asynchronous strategy which adopts non-blocking point-to-point communication between only cooperating processes. In BR methods, the main goal is to keep each process working on a reasonable amount of loads and thus prevent the occurrence of out-of-work idle states. A perfectly even workload distribution is not necessary, since the BR algorithm processes work unit one at a time and leaves the rest units standing by. It may also require heavy workload traffic to maintain such an even distribution. Therefore, a receiver initiation work-adjusting policy is set here to dynamically coordinate inter-process workload transmission when the number of boxes in a process stack is lower than some threshold. This is implemented by letting the receiver process call nonblocking Irecv to send Requests for work to all its neighbors. After receiving a request, the donor process $j$ adjusts its local workload with the receiver $i$
according to the diffusion rule

\[ u_i = D(W_j - W_i) \]  

(5.1)

where \( u_i \) is the workload-adjusting index, \( D \) is a heuristic diffusion coefficient and \( W \) is the local process work index vector. The diffusion coefficient \( D \) is set at 0.5 in this application. If \( u_i \) is positive, the donor process sends out \( u_i \) work units; otherwise, it reply with \( u_i = 0 \) to indicate that no extra workload is available. An optional upper threshold \( u_T \) is applied here to control the size of workload donation and thus reduce communication cost. This sending step is implemented by calling nonblocking Isend to initiate a sending Request and then the Wait function on the Request. The donor will thus proceed only after its response has been heard by the receiver. On the other end of the communication, the receiver process checks the status of the sending request by calling Test at the beginning of each iteration. If the request is received, work units transmitted (if there are any) will be offloaded to the local stack; otherwise, the receiver process will proceed. Therefore, communication and computation are overlapped during the execution of the BR algorithm. Through such a diffusive mechanism, processes with a large working stack can propagate workload to their lightly loaded neighbors at a small communication expense. The ADLB scheme is illustrated in Figure 5.6.

When apply ADLB to the global optimization problems, it is also important to communicate information about the upper bound on the global solution. This information is propagated here in two ways. First, the incumbent upper bound value is packed within the message of any kind of send operation (e.g., to send workload status, to send workload, etc.), Second, whenever a process obtains a better bound, either from local computation or via communication with a neigh-
Perform tests or stand by
Send work status to neighbors
Perform tests or stand by
Receive work status from neighbors
Perform tests or stand by
Send work units to neighbors
Perform tests or stand by
Receive work units from neighbors

Figure 5.6. Illustration of ADLB scheme

bor, that new bound is passed to all cooperating processes. In this way, the
local objective bound values are propagated across the entire network so that the
current best value may be quickly updated over all the processes.

The last step is to detect global termination. Without a synchronization point,
this task requires a more sophisticated solution in ADLB than in the SWS case.
As mentioned above, a popular and effective technique, Dijkstra’s token algorithm
is implemented in ADLB here to determine global termination.

5.5 Implementation with POSIX Threads (Pthreads)

In shared memory multi-core architectures, threads can be used to implement
DLB algorithms. A thread is defined as a smallest sequence of instructions that
can be scheduled independently to run by the operating system. A multi-threaded
program contains a collection of interacting or independent threads in a single process. Threads share the process resources including memory, file descriptors, file system context and signal handling, and meanwhile can run as independent entities because they maintain only the necessary resources such as stack pointer, registers, and scheduling properties for them to exist as executable code. Unlike inter-process communication in which information is “passed” between processes, inter-thread communication is realized by reading and writing shared memory (illustrated in Figure 5.7) and coordinated by thread utilities. For UNIX systems, a standardized C language threads programming interface has been defined by the IEEE POSIX standard [51]. Implementations that adhere to this standard are referred to as Pthread. The Pthread API subroutines utilized in this application can be categorized into three major groups:
Thread management functions work on threads - creating, canceling, joining, etc. This group also includes set/query routines for thread attributes.

`pthread_create` creates a new thread and makes it executable.

`pthread_cancel` cancel execution of a thread.

`pthread_attr_t` thread attribute object. Attributes include: detached or joinable state, schedule policy, schedule parameters etc.

`pthread_attr_init` and `pthread_attr_destroy` are used to initialize/destroy the thread attribute object.

`pthread_join` wait for thread termination.

Mutexes “Mutex” is an abbreviation for “mutual exclusion”. A mutex object prevents concurrent manipulation on shared data resource by more than one thread. The general idea of Pthread mutex is that only one thread can acquire the mutex at any given time. The owner thread then “locks” the mutex and accesses the piece of code that operates on the shared resource. No other thread can touch the resource until the owner thread “unlocks” the mutex. The part of code guarded by the mutex is referred to as a critical
section. Mutexes objects are thus used to prevent potential interruptions in mutually exclusive thread operations, which is a type of race condition. Mutex is one of the two fundamental thread synchronization elements. Basic mutex routines are listed below:

**pthread_mutex_t** is a mutex object.

**pthread_mutex_init** and **pthread_mutex_destroy** are used to initialize/destroy a mutex object.

**pthread_mutex_lock** is used by a thread to lock a specified mutex object. If the mutex is already owned by another thread, this call will block the calling thread until the mutex is unlocked.

**pthread_mutex_trylock** will attempt to acquire a lock on a mutex. However, if the mutex is already locked, the routine will return immediately with a busy status instead of waiting for the release of mutex lock.

**pthread_mutex_unlock** will unlock a mutex if called by the owner thread and is required after the owner thread has completed its work on protected data so that the data can be accessed later by other threads. When used in pair with pthread_mutex_lock, it builds a blocking guard which forces a calling thread to access the critical section. When called in the scope of an if statement with pthread_mutex_trylock as condition, it forms a nonblocking guard which allows a thread to bypass critical section. This is illustrated in Figure 5.8.

**Condition variables** Condition variables provide another way for thread synchronization. Condition variables allow threads to synchronize (pause or continue) based on the actual value of data. Without condition variables,
a polling function (in critical section) would possibly be frequently called to check if the condition is met. However an appropriate polling frequency may not be known beforehand, this can be quite resource consuming. Using condition variable is an efficient way to perform the same check without polling. Basic condition variable routines are listed below:

`pthread_cond_t` is a condition variable object.

`pthread_cond_init` and `pthread_cond_destroy` are used to create/destroy condition variable objects.

`pthread_cond_wait` blocks the calling thread until the specified condition is signalled.

`pthread_cond_signal` is used to signal (or wake up) another thread which is waiting on the condition variable.

Note here that operations on condition variable are mutual exclusive, and therefore both `pthread_cond_wait` and `pthread_cond_signal` must be protected by a blocking guard with using the same mutex object. An Event
object is constructed following the code from [121] to encapsulate the condition variable, the associated mutex object and the waiting status.

In multi-thread implementations, a performance degrading usage pattern may occurs when two or more threads are using unrelated data located close enough to be pushed in the same cache line. This pattern is referred to as false sharing [12]. In this application, several countermeasures, such as aligning shared global variables to cache line boundaries and storing thread specific data in thread local storage, are applied to avoid false sharing.

5.5.1 Neighbourhood Workload Pooling and Stealing (NWPS)

The NWPS scheme follows a local strategy in which each thread donates and obtains work units to/from its neighbour workload pools. The threads connected to the same pool are thus cooperative nodes, and the pools can be viewed as load traffic channels. Since all the threads operate on a shared memory space, a workload pool is simply implemented by using a standard container (std::vector in this case) and passing the reference to only the associated threads. Each thread also has its local work stack with an upper threshold $s_T$. When the local stack size grows beyond $s_T$, the thread will attempt to send one work unit to each neighbour pool. When the local stack becomes empty, the thread will try to “steal” one work unit from each neighbour workload pool. Note here that during the execution of sending or stealing operation, the work unit container is protected by mutex in the program. There are two ways to use the mutex: the nonblocking guard with calling pthread_mutex_trylock which allows a thread to bypass the load transfer stage if other thread owns the mutex lock, and the blocking guard with calling pthread_mutex_lock which forces a thread to wait until the mutex owner (thread)
release the lock. Using the nonblocking guard will reduce the waiting time, but is more likely to incur out-of-work idle states because the load balancing operation may be omitted. Applying the blocking guard will bring just the opposite effect. Only the nonblocking guard is considered here to protect workload pool. Sharing a memory space also facilitates the update of incumbent solution. The upper bound on the objective function is visible to all the threads and is protected by a nonblocking guard during update.

Each thread is assigned a condition variable which will pause the thread during the out-of-work idle state. The status of this condition variable is thus used as the work index. When a thread with an empty local stack cannot get work units from any of its neighbour pools, pthread_cond_wait will be called to pause the thread (unless the thread plays the role of termination manager). When a workload pool get donation, pthread_cond_signal will be called to awake “sleeping” threads in its neighbourhood (if there are any). Dijkstra’s token algorithm is implemented in NWPS to determine global termination with a thread being the termination manager. The termination manager checks global termination when it enters the out-of-work idle state.

The NWPS scheme is implemented following thread Pool model. A main thread will create worker threads and wait them to complete the task. Although the main thread will not further involve in the DLB process, it still will share some system resource with the worker threads.
5.6 Computational Experiments and Results

5.6.1 Test Environment

The performance of a DLB algorithm can vary significantly depending on hardware capacity and architecture of parallel computing systems. In this section, the computational experiments are carried out on a single HP Proliant SL165z G7 server node, comprising dual eight-core (16 cores) 2.3 GHz AMD Opteron processors and a 24GB RAM. Note here, although the server node is on a network-based cluster system, there is no inter-process communication between server nodes initiated in the DLB procedure. All the inter-process communications involving in the DLB algorithm are inside a single server node. In comparison to systems used in previous DLB research, for example a cluster of workstations [42], physically connected by switched Ethernet, the system used here has a lower communication cost, since MPI communication is achieved with only local memory copies when MPI processes are within a node, while extra messages passing is required across the Ethernet when communicating between different nodes. The lower communication cost is expected to reduce the effect of communication overhead on the overall performance of a DLB algorithm and thus result load balancing ability to be a more important (or even key) factor, especially for coarse grained test tasks. In the Pthread implementation, thread affinity and scheduling policies are set, so that enough number of CPU cores are provided and each core attempts to dedicate to only one thread.

5.6.2 Test Problem

This example considers a nonlinear singular optimal control problem originally formulated in [70] and later solved using interval BR method in [62]. The problem
The formulation is as follows:

\[
\begin{align*}
\min_{\theta(t)} \phi &= \int_{t_0}^{t_f} \left[ x_1^2 + x_2^2 + 0.0005(x_2 + 16t - 8 - 0.1x_3\theta^2)^2 dt \right] \quad (5.2) \\
\text{s.t.} \quad &\dot{x}_1 = x_2 \\
&\dot{x}_2 = -x_3\theta + 16t - 8 \\
&\dot{x}_3 = \theta \\
&x_0 = (0, -1, -\sqrt{5}) \\
&t \in [t_0, t_f] = [0, 1] \\
&\theta \in [-4, 10]
\end{align*}
\]

The control \( \theta \) was parameterized as a piecewise constant profile with fixed interval lengths. The problem was solved to an absolute tolerance of \( 10^{-3} \) with piecewise constant control profiles on equally spaced meshes. 4 time intervals are used in Sections 5.6.3.1-5.6.3.4. 5 time intervals are used in Section 5.6.3.5 and Section 5.6.3.6.

5.6.3 Computational Results

This optimal control problem was solved using multiple cores on a same cluster node mentioned above. During the execution of experiments, the node was dedicated exclusively to the solution of optimization problem. No processes from other cluster users are running concurrently on the node. Performance tests were taken on up to 16 cores using each of the two load balancing algorithms described above. Computation times were measured in terms of MPI wall time in MPI implementation. The performance of DLB schemes is evaluated in terms of wall time, parallel speedup (ratio of the sequential computation time to the parallel
computation time) and parallel efficiency (ratio of the parallel speedup to the number of processes used). To take account the effect of different parallel proceeding order of computation and communication, all points in result figures are based on an average over 10 runs, unless specified otherwise.

5.6.3.1 SWS vs. ADLB 1-D

In this and next sections, the optimal control problem is solved using Taylor polynomial order $q = 3$. It happens that the local solver locates the global optimum in the initialization step, and the entire BR solution procedure will thus not involve updating objective bound. This will result an equivalent amount of total workload in sequential runs and all parallel runs, since the same set of boxes (intervals) will be generated from bisections and investigations in each case. Therefore, it is appropriate to compare the performance of SWS and ADLB 1-D using this example. For parallel computation, the entire searching interval is assigned initially to one node, and other nodes are idle at the beginning. The workload is distributed in such a way that load balancing is provided solely by the DLB scheme.

The speedups obtained using SWS and ADLB 1-D, on various number of processes are shown in Figure 5.9, and the parallel efficiency curves are presented in Figure 5.10. Since a fixed total number of 31641 boxes are processed in each run, the computational results are repeatable and consistent with negligible deviations. When use 2 to 8 processes, the two schemes show just slight difference in both parallel speedup and parallel efficiency. However, SWS clearly outperforms ADLB 1-D in the case of using 16 processes. Although parallel efficiency decrease as the number of processors increases, due to rising communication overhead, the SWS
scheme can maintain an efficiency of around 87%. On the other hand, the parallel efficiency of ADLB 1-D drops to 63%. It seems that ADLB scheme can hardly be scalable to larger numbers of processes for solving this problem.

A closer look at the experiments results indicates that SWS surpasses ADLB 1-D because of a better global workload distribution. This can be seen in the gap between maximum and minimum number of work units dealt with by a single node in the network, as shown in Figure 5.11. The gap is 467 for SWS, whereas the value is 2445 for ADLB 1-D. The main reason why SWS achieves a better
workload distribution is that the most overloaded process(es) can feed multiple starving processes, and thus all the nodes on the network get work units at an early stage. The initial workload assignment setting is also a favorable factor here for using SWS. Besides, the inter-process communication are fast inside a cluster node processor and the problem is coarse-grained. Therefore, the disadvantage of using SWS, the high communication cost, becomes a less significant factor in deciding the overall parallel performance. This can be seen in Figure 5.12. The idle time in ADLB 1-D clearly outweigh the communication time in SWS.

The poor performance of ADLB on 1-D ring here does not mean that a local ADLB scheme is in general inferior to a global SWS scheme. An important aspect
of ADLB, the effect of the underlying virtual network is discussed in the next section.

5.6.3.2 ADLB 1-D vs ADLB 2-D

The effect of virtual network is investigated by solving the test problem with the same initialization procedure, as detailed above. The parallel speedup and efficiency were used again as performance measures. The 2-D mesh network is introduced here. Since 2-D mesh network requires a square number of nodes, multiple test runs were taken at four, nine and 16 processes.

The speedups obtained using ADLB 1-D and ADLB 2-D, on various number of processes are shown in Figure 5.13 and the parallel efficiency curves are presented in Figure 5.14. In all the tests, ADLB 2-D achieved higher parallel speedup and parallel efficiency than ADLB 1-D. Moreover, significant improvement is observed in the case of using 16 processes when using a 2-D mesh network. The ADLB approach on 2-D mesh also outperforms the SWS scheme, exhibiting only slightly sublinear speedup. This can also be seen in the efficiency curves. The ADLB procedure maintains a high efficiency of around 95%. Therefore, with the slightly sublinear speedup and the very high efficiency on up to 16 processes, it seems that the ADLB algorithm on 2-D network is the most promising scheme, if scale up to larger numbers of processes.

ADLB 2-D also achieved the best global workload distribution among the three schemes. As shown in Figure 5.11, the gap between maximum and minimum number of work units processed by a single node is merely 75 over a 2-D mesh network. When compared to SWS, ADLB 2-D costs significantly less time in communication, as shown in Figure 5.12. Relative to a 1-D ring, the 2-D mesh
Figure 5.13. Comparison of virtual networks: Speedup vs. Number of processes

Figure 5.14. Comparison of virtual networks: Efficiency vs. Number of processes

has a higher communication overhead due to more neighbor nodes, but results in a smaller network diameter which facilitated the work load traffic and status message exchange. It appears that this trade-off between communication cost and message diffusion favors the 2-D mesh network, especially for larger numbers of processes. The hardware architecture used also contribute to this result.

It should be noticed that ADLB 2-D achieved better performance partially due to a severely unbalanced initial workload assignment. Another test is taken here to investigate ADLB with well balanced workload initialization. In this test,
TABLE 5.1

RESULTS FOR ADLB WITH BALANCED WORKLOAD
INITIALIZATION USING 16 PROCESSES

<table>
<thead>
<tr>
<th></th>
<th>1-D</th>
<th>2-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel speedup</td>
<td>15.74</td>
<td>15.75</td>
</tr>
<tr>
<td>Parallel efficiency (%)</td>
<td>98.36</td>
<td>98.44</td>
</tr>
<tr>
<td>Percentage of idle time (%)</td>
<td>0.37</td>
<td>0.23</td>
</tr>
<tr>
<td>Percentage of communication time (%)</td>
<td>0.10</td>
<td>0.07</td>
</tr>
<tr>
<td>Largest gap of units processed</td>
<td>2934</td>
<td>2851</td>
</tr>
</tbody>
</table>

Each of 16 processes is given the same initial interval so that the loads distribution are precisely even at the beginning. A fixed total number of 506256 boxes are processed in each run. The results are presented in Table 5.1. 2-D mesh network merely offers slight improvement over 1-D ring in this case. This indicates that the message diffusion is a less important factor when no nodes suffer the initial “starving” state.

Since ADLB 2-D exhibits better performance, this scheme is selected as the candidate for further investigation on a more challenging scenario in which a good initial guess is not available.

\(^{1}\)The gap between maximum and minimum number of work units processed by a single node.
5.6.3.3 Worst-case Scenario Test

In general, a good objective bound may not be obtained in the initialization step or at an early stage of a BR procedure. In this section, the performance of the ADLB 2-D scheme is tested on a worst-case scenario in which a good objective bound will be found in a late stage in the sequential runs. In order to create such a scenario, the local optimization solver is turned off in the initialization step; the maximum double precision number ($1.79769e + 308$) is set manually as the initial guess; the entire searching space is divided into 16 equal subspaces on the first parameter; the subspaces are piled in one node with the one containing the global solution placed at the bottom of the work unit stack. Multiple test runs were taken at four, nine and 16 processes.

The resulting parallel speedups are shown in Figure 5.15. Besides average speedup, maximum and minimum speedup are presented in each case as well. Sublinear speedup is observed only over the test of 4 processes. All the runs over 9 and 16 processes show superliner speedup. Both superlinear and sublinear speedup are possible because of the objective bounds propagation and work units transport, which may cause subintervals (boxes) to be dealt with (bisected, reduced or discarded) earlier or later than in the sequential case, i.e. the total amount of workload in the parallel case is different from that in the sequential case. The speedup results vary significantly from run to run, because improved objective bounds are found and passed at different time in each case. When using more processes, it is more likely that the subinterval enclosing the global optimum will be investigated at an early stage and better speedup may thus be achieved. Such speedup anomalies are not uncommon phenomena in parallel BR procedure.
5.6.3.4 Task Granularity

In this section, we investigate the impact of task granularity on the performance of ADLB 2-D scheme. The task granularity is adjusted by increasing the Taylor polynomial order to $q = 5$. It costs on average 170% more time per task than using $q = 3$. Since the larger Taylor polynomial order $q$ also lead to reduced propagation of overestimation (better solution bounds) in each task, a less total number of 30142 boxes are processed in each run. Other initialization settings are the same as detailed in Section 5.6.3.1. The parallel speedup and efficiency curves are presented in Figure 5.16 and Figure 5.17 respectively. Increasing the Taylor polynomial order does not have significant impact on the performance of ADLB 2-D scheme. ADLB-2 can still maintain a parallel efficiency at 94%.

5.6.3.5 NWPS 1-D vs NWPS 2-D

In this and next sections, the performance of Pthread implementation is investigated with using the same initial workload assignment as described in Section 5.6.3.1. We set Taylor polynomial order to $q = 5$ and use a control profile of 5 time intervals. A total number of 532703 boxes are processed in each test run.

Figure 5.15. Speedup achieved by using ADLB 2-D scheme on worst-case scenario
The sequential runs are taken by letting a main thread create a worker thread. The speedups obtained using NWPS 1-D and NWPS 2-D, on various number of threads are shown in Figure 5.18, and the parallel efficiency curves are presented in Figure 5.19. In all the tests, NWPS 2-D achieved better results than NWPS 1-D. In the test of 16 threads, significant improvement in workload balancing is observed again when using a 2-D mesh network. As shown in Figure 5.20, the gap between maximum and minimum number of work units processed is significantly reduced and thus less time is spent in waiting during idle state as shown in Fig-
In the Pthread implementation, the communication cost appears to be negligible due to the fact that communication here is merely to access the shared memory. Therefore, the trade-off between communication overhead and work units diffusion again favors the 2-D mesh network, especially for larger numbers of threads. It is also observed that the total computation time of NWPS 2-D can be up to 20% more than that of NWPS 1-D or the sequential case. This indicates that contention of certain system resource (e.g. light-weight processes) may exist and the computation time includes content switch time. The contention occurs more when use 2-D network since nodes are always busy during the execution.

The effect of the virtual network appears to be of the same trend as in the MPI implementation. 2-D network leads to higher performance. More comparison between MPI and pthread implementations is given in the next section.

![Figure 5.18. Comparison of load balancing schemes: Speedup vs. Number of threads](image)
Figure 5.19. Comparison of load balancing schemes: Efficiency vs. Number of threads

Figure 5.20. Maximum and minimum number of work units tested on a single node out of a 16 nodes network

Figure 5.21. A distribution of elapsed time of Pthreads implementation on two virtual networks
5.6.3.6 MPI vs. Pthread

In this section, we perform the same test (as in Section 5.6.3.5) using the MPI implementation with ADLB 2-D scheme. The sequential runs for MPI are taken here using only a main program thread and cost 30% less time than that in the Pthread implementation. The wall time obtained on various number of nodes are shown in Figure 5.22. When applied on the same 2-D network, MPI outperforms Pthread in all the runs. The parallel speedup and efficiency curves are presented in Figure 5.23 and Figure 5.24 respectively. In the case of 16 nodes, the MPI implementation can maintain a 92% parallel efficiency, while the value drops to 75% using the NWPS scheme. This difference is due to extra cost of system threads management and contention of system resource in Pthread rather than the load balancing ability.

![Figure 5.22. Comparison of MPI and Pthread: Wall time vs. Number of nodes](image)

Figure 5.22. Comparison of MPI and Pthread: Wall time vs. Number of nodes
Figure 5.23. Comparison of MPI and Pthread: Speedup vs. Number of nodes

Figure 5.24. Comparison of MPI and Pthread: Efficiency vs. Number of nodes
CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Rigorous Global Optimization for Dynamic Systems Subject to Inequality Path Constraints

We have demonstrated here a new approach for the rigorous, deterministic global optimization of dynamic systems subject to IPCs. While various methods have been proposed for the deterministic global optimization of unconstrained dynamic systems, the approach presented here appears to be the first to address this constrained problem. The method employs a branch-and-reduce approach based on use of interval analysis and Taylor models. Unlike previous optimization methods for problems with IPCs, which enforce the IPCs only at the level of the discretization used in the problem formulation or in the numerical routines used, in this new approach the IPCs are guaranteed to be satisfied continuously, rather than just at discrete points in time. Since the IPCs may be safety related, a conservative approach is followed in which the search for the global optimum is restricted to a space in which continuous satisfaction of the IPCs is rigorously guaranteed, and an $\epsilon$-global optimum within this space is determined.
6.2 Robust Design and Global Optimization for Nonlinear Dynamic Systems

We have demonstrated here a new solution framework for robust design and global optimization of nonlinear dynamic systems. Since this method is derived from our global optimization approach, it inherits key features such as rigorous constraints satisfaction and guaranteed $\epsilon$-global optimality for inner layer solution. The use of a bi-level branch-and-reduce is demonstrated to be able to reduce redundant work and lead to efficient decision. A conservative approach is applied here. For robust design problem, the solution is a set of operational (decision) subregions in which satisfaction of all the constraints are rigorously guaranteed over the entire uncertain parameter space. For robust optimization problems, the search for global solution is confined in the feasible regions and a global optimum is determined with an $\epsilon$-convergence guarantee for both the inner (max) and outer (min) optimization problem.

6.3 Dynamic Load Balancing

We have presented how dynamic load balancing strategies can be used within an interval BR algorithmic framework on a multi-core cluster server node. Two parallel implementations considered here are Pthread and MPI. In both versions, virtual network appears to be an important factor in determining the overall performance. Among all the DLB schemes considered, the asynchronous diffusive load balancing on 2-D network (ADLB 2-D) leads to the best performance. The results of using ADLB on 2-D mesh in the global optimization problem with best case initialization have shown that the parallel scheme is highly scalable up to 16 nodes. In the worst case scenario, the parallel BR algorithm is able to consistently achieve superlinear speedups. Therefore, the ADLB 2-D is the most promising
scheme to address tasks of a similar or larger granularity. Such tasks are often encountered when the verified Taylor model approach is applied to solve dynamic optimization problems.

Although the DLB schemes are designed here to parallelize the interval BR procedure, it should be noted that the schemes are in general applicable to a wide range of search tree problems in process engineering field.

6.4 Future Work

We have demonstrated that interval analysis and verified method can be utilized to construct rigorous solution procedures for dynamic optimization and design problems and concurrent parallel programming techniques can be applied to improve the computational performance. However, there still exists a wide range of dynamic design and optimization applications which in practice will be very difficult to solve using rigorous approach. To tackle these challenging problems, it requires continuing improvement in mathematical methods, computational algorithms and software design. According to our experience, improvements may be made in following directions,

1. Adaptive tuning. In Chapter 3 we used a technique for the automatic tuning of the algorithm parameters. This method is based on a stochastic sampling of the search domain and determines parameter values that are then applied over this entire domain. We anticipate that the computational performance of the global optimization algorithm can be improved by using an adaptive tuning, in which algorithm parameters may differ in different parts of the search domain, based perhaps on local sampling or current search box size.

2. Sensitivity analysis. In optimization applications, it is common that objec-
tive function and/or constraint function are more sensitive to changes in some parameters than the others. Sensitivity analysis can provide this information and lead to wise bisection decision. Note that sensitivity analysis will bring extra computational cost. A compromise approach may be used. That is, only perform sensitivity analysis until the algorithm is proceeding to certain stage. (e.g., when the function bound obtained is smaller than a tolerance.)

3. Fine grain parallelism. As introduced in Chapter 5, there are multiple opportunities for implementing parallel computation in our optimization algorithms. Ideally, we want to realize parallel computation at all possible levels. Finer grain scheme, such as parallel interval arithmetic and Taylor model operations may be applied within a dynamic load balancing framework when suitable hardware is available.
BIBLIOGRAPHY


