PRACTICAL OPTIMIZATION OF SIMULATIONS: COMPUTATION AND TOOLS

By

Krung Sinapiromsaran

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

2000

Abstract

The main topic of this thesis is solving simulation optimizations using a deterministic nonlinear solver based on the sample-path optimization concept. The simulation function is considered as a black box that deterministically returns the exact output for the same input value. The gradient-based nonlinear solver finds a local optimal based on the function and gradient evaluation of the sample path simulation function. The simulation output is used for the function evaluation while the derivative of a quadratic model is returned for the gradient evaluation. We locally build a quadratic model from the surrounding simulation points using a least squares approximation. This scheme does not require the modification of the original simulation source code and can be carried out automatically.

Due to the large number of simulation runs, the high-throughput computing environment, CONDOR, is used. Simulation computations are then distributed over heterogeneous machines which can be executed on entirely different computer architectures within the network. Additionally, a resource failure is automatically handled using the checkpoint and migration feature in this CONDOR environment. We implement a Master-Worker condor PVM server to avoid CONDOR scheduler waiting period overhead.

At the end of the thesis, we show how to solve a nonlinear programming problem using a primal-dual formulation. The first and second order derivative models of the Lagrangian function are built using an automatic differentiation that is supplied by a modeling language. In addition, the original nonlinear objective function is incorporated into a Fischer-Burmeister merit function to guide the mixed complementarity solver to find the optimal solution of a nonlinear program.

Keywords: Simulation optimization, gradient based optimization, sample-path assumption, modeling language, nonlinear program, high-throughput computing environment, Complementarity problem, automatic differentiation.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Professor Michael C. Ferris, for his invaluable time and advice. I would also like to thank members of my final defense, Professors Robert R. Meyer and Professor Stephen M. Robinson for revising my thesis, Professor Olvi L. Mangasarian and Professor Mary K. Vernon for their valuable time to read and criticize this work.

I would personally thank Todd S. Munson and Qun Chen for their helps in many ways. I would like to thank all the CONDOR teams at UW-Madison for dedicating their time to solve my CONDOR related problems, especially Sanjeev Kulkarni who spent his time to show me the Master-Worker program. Thank to people in the CS department that manage the computer resources. This research was partially supported by National Science Foundation Grant CCR-9972372 and Air Force Office of Scientific Research Grant F49620-98-1-0417.

Finally, I would like to convey my sincerest regards to my parents for their love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Many practical engineering problems give rise to intractable mathematical formulations. The complex nature of the system or the real-world process motivates researchers to employ simulation modeling as an alternative method to draw a conclusion about the system behavior. A simulation model is used to describe and analyze both the existing and the conceptual system, sometimes helping in the design of improvements for the real system. Good summaries on simulation components and the advantages and disadvantages of using simulation models can be found in [75, 8]. Many usages of a simulation model can be found in [50] for manufacturing applications, in [85] for the aviation industry, in [57] for chemical analyses, and in [80] for the health care industry.

The popularity of simulation models comes from their use as decision making aids for determining possible outputs under various assumptions. The common use of simulation models is in goal-driven analyses which determine values of the decision variables to allow the model to achieve specific goals. This goal-driven simulation is used in many fields of economics, business and engineering. Moreover, the decision variables are typically constrained by other relationships, for example, budgetary or feasibility restrictions. These constraints together with a goal give rise to an optimization model using the goal as an objective function.

There are two classes of goal-driven simulation optimization. The first class is the level crossing which tries to identify the decision variables of a simulation that produces the simulation output with the specified value, d. We can cast the optimization of this class of problems as $\min||S_r(x) - d||_p$ where $S_r(x)$ is the simulation function, d is the target value and $||\cdot||_p$ is the p-norm measure function, usually the Euclidean norm. The second class is straightforward optimization which tries to determine the optimizer of the decision variables from the simulation. This problem can easily be cast as a mathematical programming problem.

From the optimization perspective, the simulation model or function is simply a function that takes input values and derives one or more output values. In reality, most optimization algorithms also rely on the first order or second order derivative of the simulation function. Due to random effects and the stochastic nature of the simulation, the exactness of the gradient of the simulation function may misguide the optimization algorithm to search an entirely wrong direction. More reliable methods to estimate derivatives must be used.

In addition to the noise of the simulation, the long running time of a simulation model is also an unfavorable situation. Even though the use of the simulation model is cheaper than building the real system, it is very time-consuming to run and analyze. One simulation run may consume hours of execution time on a single computer. Furthermore, multiple simulation runs are required in order to reliably estimate a steady state simulation function or estimate its gradient. Many methods have been used to reduce the execution time of this problem, such as infinitesimal perturbation analysis which approximates the gradient evaluation using one simulation run [31, 42, 60, 61] or the score function method (also called Likelihood Ratio method) [69, 68].

The correctness of the simulation model [72, 70] is an important ingredient for an optimization problem involving simulation. The verification of the simulation program ensures that the computerized simulation program is performed accurately, and a validation of the simulation model determines if a simulation model is an acceptable representation of a real system. See [47] for testing the validity of the model using a statistical method. We will assume throughout that the simulation model is correct before we start optimizing it.

1.2 Simulation Optimization

The optimization of simulations is a challenging optimization problem that researchers have attacked for many years from the 1950's until the present day [46, 78, 2, 73, 6, 9, 33, 5, 40, 60, 7, 74, 66, 61, 48, 76, 77, 58, 32, 71, 55, 79, 59, 39, 41]. We can categorize the methods for solving the simulation optimization into two categories.

The first category is comprised of methods which optimize a deterministic simulation model. These methods include a meta-model optimization [71, 59], or a sample-path optimization [66, 39, 60, 61, 26] (that we will use in this thesis based on a different gradient evaluation procedure). A meta-model method considers the steady state simulation model as a complicated formula that is approximated by a nonlinear function. Then the optimization process is applied to this nonlinear function to find the best optimum solution. The simulation model assumes a mathematical formula as

$$y = S_r(x)$$

where y is a vector of a system response, x is a vector of decision variables and S_r is a simulation model. The meta-model approximation of this simulation is then defined as

$$y = f(x, \theta) + \epsilon$$

where x is a decision variable, θ is a parameter vector to be estimated from the algorithm and ϵ is the error. Santos and Porta Nova [71] use the least squares method to estimate this meta-model and validate the model using regression analysis. The correctness of this method relies on the exactness of the meta-model approximation and the optimization algorithm. Another method is a goal-driven simulation algorithm [59] which solves the special form of this meta-model. By considering a simulation model as one of the nonlinear constraints, Morito et al. [55] suggest an iterative method that uses a variation of the cutting-plane method to find the optimal solution of the simulation model under constraints. They apply a deterministic optimization method to an objective function and nonlinear constraints except a simulation model. They treat a simulation model as an oracle to verify the feasibility. So if a simulation returns an unacceptable result, they reduce the searching domain by adding mathematical constraints. However, this method is hard to generalize and is likely to be poorly behaving for a standard simulation optimization problem.

In this thesis, we use the sample-path optimization method which is described in section 1.3 that relies on a deterministic optimization algorithm solving the long simulation model based on the sample-path optimization assumption. There are two different approaches that can apply to this scheme. The first approach is a derivative-free optimization [14, 17, 83, 84, 51] which does not rely on the derivative of the actual simulation run. Conn and Toint [14] apply the derivative-free technique by using an interpolation set that defines a (quadratic) polynomial model and retrieve a gradient direction from this polynomial model. They integrate their method with a trust region algorithm and solve problems from the CUTE [11] and Hock and Schittkowski [43] test sets with success. We do not use this technique because it does not integrate well with constrained optimization techniques and existing mathematical algorithms such as CONOPT. Dennis and Torczon [17] apply the pattern search algorithm which totally avoids computation of the derivative of the objective function and explores a lattice around the current point. Under a regularity assumption that the objective function is continuously differentiable, they show that their method converges [83, 84] to a first-order point. In addition, if all constraints are linear [51], then a variation of this method is guaranteed to converge to the Karush-Kuhn-Tucker (KKT) solution. However, this method may take many function evaluations to converge. The second approach is a gradient based algorithm of the sample path optimization [37, 38, 61, 35, 36, 26] that relies on the derivative computation of a simulation function. We will use this method in the thesis which will be discussed in in section 1.3 in detail.

The second category is comprised of methods that optimize the stochastic simulation model directly or the simulation model with noise. These methods include stochastic optimization [63, 33, 79, 74], or heuristic optimization approaches such as Genetic algorithms [7, 41], Simulated annealing [40], Nested Partitions [77, 76, 58], or the Tabu search method [32]. Even though these methods converge to the solution in the presence of noise, they are usually slower than deterministic mathematical programming methods.

Stochastic approaches estimate the optimal solution by generating a sequences $\{x_n\}$ where

$$x_{n+1} = x_n + a_n g(x_n, \theta)$$

for all $n \geq 1$, where $g(x_n, \theta)$ is an estimate of the gradient of the simulation function at x_n. The sequence a_n has infinite sum with a finite second moment. Under appropriate conditions, the sequence $\{x_n\}$ is guaranteed to converge to the optimizer with probability one.

Heuristic approaches use the simulation output to define a merit function that determines the goodness of that point. They use some randomness to select a group of candidate points and apply operations on those points. If the selected points do not improve the objective function of the problem, then backtracking or drawing of new candidate points will be used. There is no guarantee of the convergence of these methods except in the discrete case [76, 58] which is equivalent to generating all possible outputs. For a good coverage of the stochastic optimization method, see [16].

1.3 Sample-path optimization

Suppose we are given an extended-real-valued stochastic process $\{S_n\}$ that is a function of a decision variable $x \in I\!R^p$ with the property that it almost surely converges to an extended-real-valued function S_∞. For example, S_∞ is the steady-state function of a simulation while $\{S_n\}$ is a simulation of length n. Assume that S_n and S_∞ are defined on a common probability space $(\Omega, \mathcal{F}, P)$. A sample path for these stochastic processes obtained by fixing a particular ω and is defined by $\{S_n(\omega, \cdot)\}$ and $S_\infty(\omega, \cdot)$, respectively. Note that each of these functions is deterministic, so we can compute the value of S_n based on this sample. Since S_∞ may be intractable and hard to compute, we aim to find a minimizer of $S_\infty(\omega, x)$ using $S_n(\omega, x)$. Generally, the minimizer of $S_n(\omega, x)$ may not converge to a minimizer of $S_\infty(\omega, x)$ unless the sample-path assumption [66] that we now define is satisfied.

Before stating the sample-path assumption, we need to introduce the definition of a lower semi-continuous and proper function and the concept of epiconvergence [4, 44].

An extended-real-valued function defined on $I\!R^p$ is said to be proper if and only if $-\infty$ does not belong to its range and its value is not equal to $+\infty$ everywhere. The extended-real-valued function, S is said to be lower semi-continuous at $\hat{x} \in I\!R^p$ if and only if for any sequence $\{x_n\}$ in $I\!R^p$ converging to $\hat{x}$, $S(\hat{x}) \leq lim_{n\to\infty} S(x_n)$ if $lim_{n\to\infty} S(x_n)$ exists.

We say that a sequence of extended-real-valued functions, S_n, epiconverge to a function S if

1. For every $\{x_n\}$ in $\mathbb{R}^p$ that converges to $\hat{x}$, $S(\hat{x}) \leq \liminf_{n\to\infty} S_n(x_n)$
2. There exists a sequence $\{y_n\}$ in $\mathbb{R}^p$ converging to $\hat{x}$ such that $S(\hat{x}) \geq \limsup_{n\to\infty} S_n(y_n)$

Theorem 1 (Robinson, 1996) *Suppose*

1. *The functions $\{S_n(\omega, \cdot)\}$ are proper, lower semi-continuous and epiconverge to $S_\infty(\omega, \cdot)$ with probability one*
2. *$S_\infty(\omega, \cdot)$ is proper with a nonempty compact set of minimizers.*

Then for large n, a set of minimizer of $\{S_n(\omega, \cdot)\}$ is nonempty and compact and a point in this set is within a small distance from some point in the set of minimizers of $S_\infty(\omega, \cdot)$ with probability one.

Our main usage of the sample-path optimization is to apply a deterministic optimization algorithm to the simulation program for a fixed simulation length based on the same stream of random numbers. Since most simulation implementations use pseudo random number generators from a fixed seed, the simulation function with the same random seed will generate the same output values. Hence, the simulation is deterministic and we can apply a deterministic optimization algorithm to find the minimizer of this function which is close to some minimizer of the steady-state function. In the next section, we will show how to estimate gradient of the simulation function based on the local quadratic model that can be used in the gradient-based deterministic optimization algorithm.

1.4 Gradient evaluations for a simulation function

To guide gradient based optimization to solve a simulation optimization problem, different methods for gradient computation have been used; finite differences, infinitesimal perturbation analysis [42], automatic differentiation [35, 36], or the derivative of a simple polynomial fitting model as in this thesis.

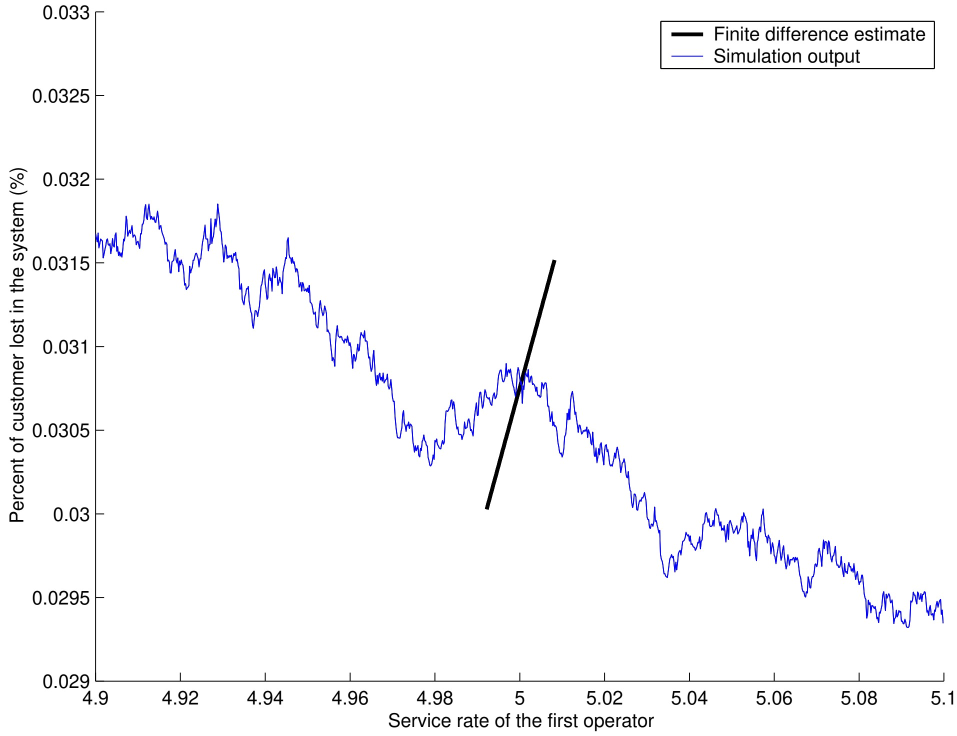

Figure 1: Noisy simulation function and the its derivative at 5.0 using a finite differences.

The first method, finite differences, is very general and easy to implement. For n decision variables, the method relies on the simulation computation of at least $n+1$ simulation points. The numerical accuracy of the derivative results from the small distance between these points. This causes difficulty in handling the derivative accuracy. The method is not suitable to deal with noisy functions because the estimation of a noisy function could return derivatives that are completely erroneous. Figure 1 shows an example of the derivative computation at 5.0 based on the finite difference method. This is a graph of the service rate of the first operator against the percentage of customers lost in a telemarketing simulation. From this figure, the derivative approximation at 5.0 shows an increasing trend of the function to the right. However, the obvious minimal solution lies on the right side of the graph. The final optimal solution using this method is highly dependent on the starting point chosen.

The second method, infinitesimal perturbation analysis (IPA), uses a sample path simulation run to collect the gradient evaluation of a simulation output. This method requires a modification of the simulation source code to incorporate the gradient computation. Even though the method is more efficient than finite differences, it inherits the same problem regarding noisy functions because the derivative result from the IPA method is equivalent to the finite difference method.

FORTRAN

```
if(x .lt. y) then
   z = x
else
   z = y
endif
```

Figure 2: FORTRAN code example for `z = min(x,y)` using branching.

The third method, automatic differentiation uses the evaluation of the underlying function to compute the derivative code. The method constructs and modifies the evaluation function tree. For analytical functions, this method can compute derivatives of a long and complex function expression faster than a symbolic differentiation because of repeated algebraic formulae occurring in the expression. Due to complex nature of a simulation model, it is implemented using a typical programming language such as FORTRAN or

C which contain some discrete variables and branching conditions, as shown in Figure 2. The problem in using automatic differentiation stems from these two components. The first component is that the derivatives of any discrete variables are not mathematically well defined. For example in M/M/1 queue, an average waiting time, w, is computed as the total waiting time tw divided by m customers. Clearly the derivative of $\frac{\partial w}{\partial m}$ is not defined when m is finite and integer. The second component, the branching conditions, causes the evaluation tree to return different values from small changes that trigger the branching condition. This leads the automatic differentiation to produce a sharp derivative change. Figure 2 shows the sample FORTRAN code that computes the value of `z`. Note that the gradient of `z = min(x,y)` with respect to `x` or `y` is not well defined. Additionally, this method requires the simulation source code which we try to avoid in this thesis.

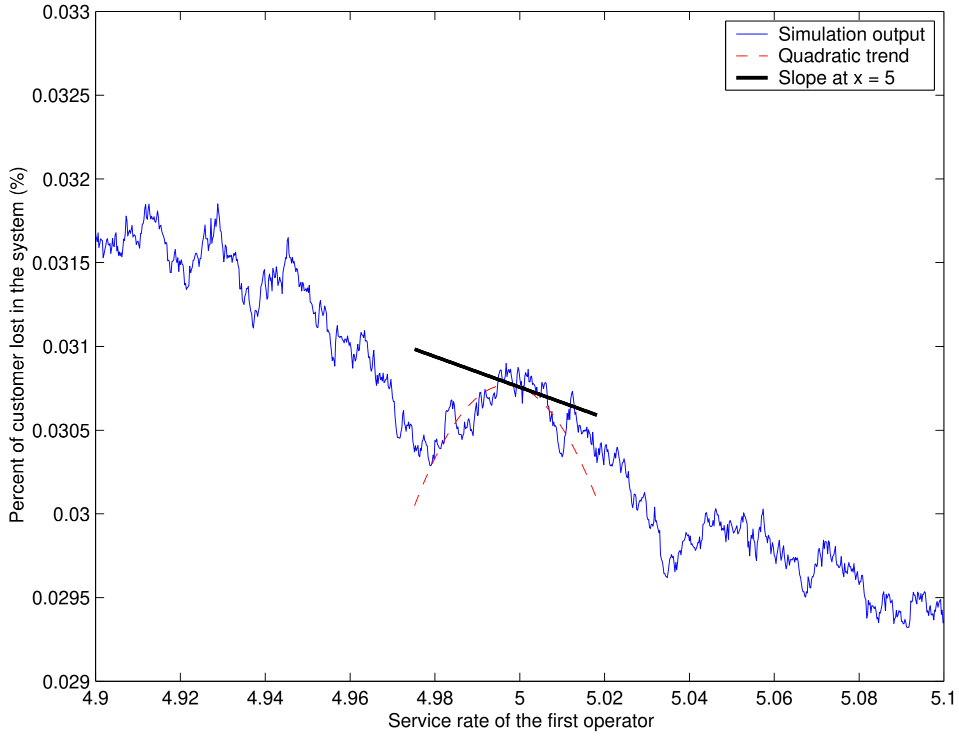

Figure 3: Noisy simulation function using a local quadratic model for gradient approximation at 5.0.

Unlike the second and the third methods that require access to the simulation source code, our method relies solely on executing the simulations of surrounding points and building up a (small-scale) quadratic model which is smooth and differentiable similar to the derivative free method. While this approach may be inefficient because of large computation of the simulation runs, the derivative approximation is more reliable and the computation can be carried out entirely automatically. Figure 3 shows the same example as figure 1 using the locally fit quadratic model. The gradient approximation at 5.0 shows a negative slope which guides the optimization algorithm to search for the minimal point to the right of the graph.

1.5 Overview of the quadratic simulation optimization algorithm

Figure 4 shows our scheme to solve a simulation optimization. A practitioner supplies a nonlinear model written in the GAMS modeling language [12] and a simulation module written in traditional programming language such as FORTRAN or C as an external routine. A nonlinear solver is selected using the modeling language statements. The simulation optimization problem is solved by executing the model in the modeling language system. The modeling system first constructs an internal nonlinear model and passes the information to the nonlinear solver. Except for the function and gradient of the external equation, all function and derivative evaluations are supplied by the modeling language using its internal evaluation and automatic differentiation as appropriate. For the evaluation of the external function call, the modeling language system gives control to the quadratic simulation module which calls the simulation module for function computation. However, the derivative computation is different. The module constructs the quadratic model using surrounding simulation points computed by the simulation module. It refines the quadratic model using a statistical test. If the first-order condition is satisfied or the termination criteria is met, then the nonlinear solver returns the result to the modeling language system. The modeling language system then reports the final result to the modeler.

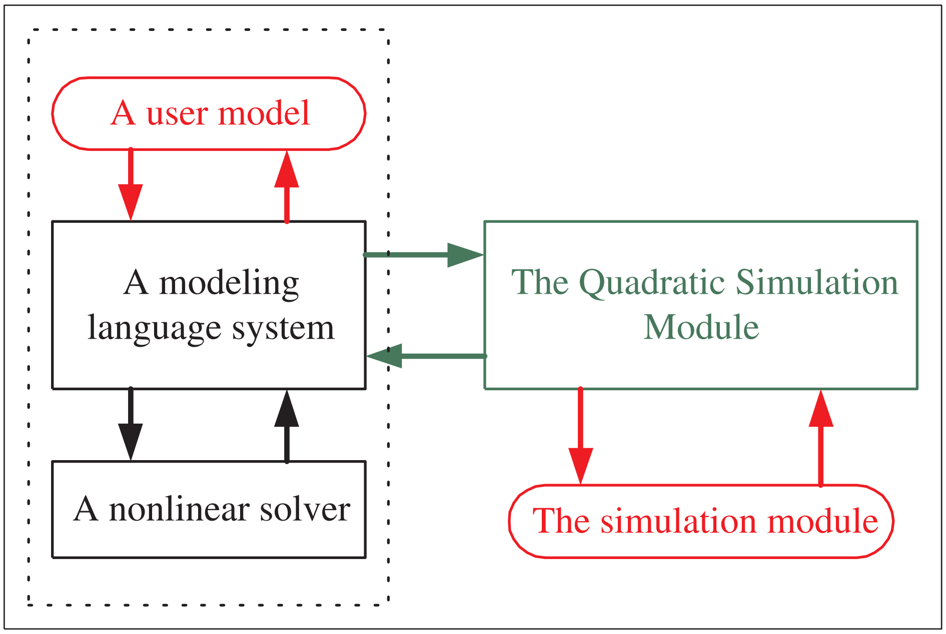

Figure 4: Overview of the simulation optimization solver.

1.5.1 Quadratic model

In this thesis, we address a practical approach for solving the simulation optimization of continuous decision variables based on sample-path optimization. We allow the optimization model to be formulated as the nonlinear program in the GAMS modeling system [12]. The simulation function is provided essentially as a black-box module to the GAMS model as an external equation. Our approach exploits state-of-the-art (gradient based) optimization solvers, rather than stochastic or heuristic ones.

Our approach is modular because it can easily apply a new state-of-the-art nonlinear programming solver without recoding additional information nor programming. This is accomplished by a solver link and the dynamic linked library routines within a modeling language system. To deal with a stochastic function such as a simulation, we use a model building technique that relies on the surrounding points which are randomly and independently generated. For a linear model of n independent variables, this technique requires at least $n + 1$ sampling points which is the same requirement as the finite difference method. For the quadratic model of n independent variables, it requires at least $n(n+1)/2 + n + 1$ for a symmetric quadratic model and $n^2 + n + 1$ for a non-symmetric quadratic model. For higher order polynomial models of n independent variables, its requirement of sampling points grows in the order of n^3. We use a quadratic model because of its nonlinear nature and its smaller requirement for the number of sampling points. In addition, a higher order polynomial model can cause the over-fitting problem which captures simulation noise in the model.

1.5.2 Statistical test of quadratic model

After the simulation runs are complete, we build a small-scale quadratic model using least squares approximation. The aim is to find the best quadratic fitting model of the simulation points that exhibits a similar derivative to the gradient of the simulation function. The least squares approximation of finding the quadratic fitting model is equivalent to performing a linear regression model on the coefficients of the quadratic model using the simulation points as the regression data. The analysis of regression can then be used to determine the best quadratic fitting model.

We use the coefficient of determination, R^2, to test the fitness of our quadratic model to the simulation data. This R^2 can be interpreted as the ratio of the regression variation and the total variation in the data points where the regression variation is the difference between the total variation and the error variation.

If R^2 is close to 1, then the regression variation is approximately equal to the total variation. In another words, the error variation is close to zero. We can use the quadratic model to predict all simulation points so that the derivative of this quadratic model is close to the gradient of the simulation function within that neighborhood. If R^2 is close to 0, then the total variation is close to the error variation. That means the quadratic model could not be used to predict any of the data points within that neighborhood.

Because of the stochastic behavior of the simulation, the quadratic model always has error variation even though it perfectly fits the trend of the simulation function. Instead of rejecting the model, we derive an additional statistical test to be able to accept this type of quadratic model.

In this thesis, we propose that the acceptable quadratic model in the neighborhood of the point can be 1) a quadratic model that shows acceptable fitness, i.e. the R^2 is close to 1, or 2) a quadratic model with white noise error. The white noise error is defined as the error distribution that is normal with unknown mean and unknown variance. This can be determined using goodness-of-fit test statistics.

Goodness-of-fit test statistics[81] relate to the problem of determining whether the samples $x_1, x_2, ..., x_n$ are extracted from a population with known distribution $F(x)$. Generally, the Chi-square test is used because it can be easily adapted to any distribution, both discrete and continuous. Because we are interested in the goodness-of-fit test statistics with the normal distribution $F(x)$, a more powerful test such as the Cramér-von Mises statistic[82] (herein termed the W^2 statistic), can be used. The W^2 statistic is designed to test the null hypothesis that the distribution of data fits a general distribution such as a normal or an exponential. When the null hypothesis is rejected, then with high probability the error distribution reveals that some simulation points (extreme points) do not fit with the current quadratic model. In this case, the quadratic model is not acceptable. Since we do not plan to fit the simulation using a higher order polynomial model, we re-fit the quadratic model based on smaller radius which excludes those extreme points.

1.5.3 Identifying outliers

For noisy functions, the quadratic model using the least squares approximation does not always fit the function. The error distribution of the fit may exhibit a non-symmetric distribution that is caused by some extreme values. In order to fit the quadratic model with this noisy function, we want to exclude these points from our quadratic model building.

To determine extreme points or outliers from the simulation data, we use a skewness measure of the distribution shape. If there are outliers or extreme points in the sample, then the shape of the distribution is not symmetric with respect to the mean or the distribution skews. A distribution is said to be skewed to the right if the mean is larger than the median and it is said to be skewed to the left if the mean is smaller than the median. The coefficient of skewness [1] can be used to determine the lack of the symmetry in data. It is computed as $\frac{\sum_{i=1}^{n}(x_i-\bar{x})^3}{(n-1)\times s_e^3}$, the ratio of the expectation of the third power of the sample from its means to the third power of the standard deviation. If the data is distributed symmetrically from the left and the right of the center point, then the skewness is zero. For example, the normal and uniform distribution have zero skewness. If the distribution is skewed to the left, the skewness is negative. The positive skewness indicates that the distribution is skewed to the right.

The skewness can be used to determine the group of extreme points or outliers that we need to exclude from our model building. By reducing the radius, we can exclude some undesirable point from our model. The strategy is to obtain the largest radius so the model building can use as many points as possible.

We sort all points according to their values and group them in the fixed number of blocks. Therefore, we partition our points into blocks of data which are corresponding the block of histogram. If the coefficient of skewness is negative or the distribution is skewed to the left, the highest block is discarded because it contains extreme points. However, if the derivative point is in this block, we rebuild the quadratic model within this block. Otherwise, we discard the block by computing the largest radius that all points in that block are excluded.

1.5.4 Parallel computation of quadratic model

The quadratic model still requires large amounts of computational power for the simulation computations. Nevertheless, the simulation points can be computed in parallel using a high-throughput computing environment, such as CONDOR [54, 21].

CONDOR is a resource management system that utilizes the computational resources within the network using a ClassAd design. The ClassAd design is made up of the resource agents who advertise their resource in a resource offer ad, the customer agent who requests the machine to run a task, and the matchmaker who identifies the resource matching between the resource agents and the customer agent. After the match has been made, the customer agent needs to send the description of the task to the resource agent which is done using the file system. CONDOR also supports efficient mechanisms to access data files of the customer agent from the resource agents and attempts to optimize the processing time of executing jobs within the network. Because of the flexible design of CONDOR, the customer agent can request multiple architecture resources to execute the task.

To take advantage of the heterogeneous machines within the CONDOR pool, we compile our simulation module on different operating systems on different machines. The simulation code is re-linked to the CONDOR library to be able to utilize the checkpoint and migration facilities. Internally, this maintains a trace of the execution of the task so that another machine can restart the execution of the program when the task has been interrupted. Each simulation run may execute on entirely different machine architecture within the network.

We also use a Master-Worker paradigm to alleviate this computational problem. The Master-Worker server runs as a separate process along with the optimization solver. Both use the file system to exchange information. This significantly reduces the overhead of waiting and rescheduling time that occurs within the CONDOR environment.

1.6 Nonlinear programming solver

In this thesis, we also use a model building technique to solve nonlinear programs based on the primal-dual formulation [20] of the Lagrangian function using second order information. The first-order conditions of the original NLP model are cast as a mixed complementarity model which is solved using the Newton type complementarity solver, PATH[18].

The first order derivatives of the original nonlinear model are required for constructing the mixed complementarity model while the second order derivatives are employed by the mixed complementarity solver. In this situation, nonlinear functions are defined within the modeling language system that provides the the first and higher order derivative using automatic differentiation. Because the computation of the automatic differentiation uses the function evaluation tree, the numerical results are guaranteed to be accurate up to the machine precision.

The mixed complementarity solver, PATH, uses a merit function to guide and identify the solution point. To guide the mixed complementarity solver to find the desired optimal solution, we implement a new merit function as a weighted sum of a residual function for error in the first-order conditions and the nonlinear objective function.

1.7 Scope of the thesis

The objective of this thesis is to develop a general and practical simulation optimization solver that requires no modification of the simulation source code and easily adapts to new state-of-the-art nonlinear programming solvers. The program could be used to explore and extend a variety of simulation optimizations based on the same simulation module. In addition, the technique of model building can be applied to solve a nonlinear program using a state-of-the-art mixed complementarity solver. The major contributions of this research are

1. Development of the gradient based simulation optimization using the quadratic fitting model;

2. Development of the parallel computation of quadratic simulation approximation

solver using the resource management system, CONDOR;

3. Development of adaptable simulation optimization model in the modeling language that reuses the same simulation module without modifying the simulation source code.

4. Combination of different programming environments using a dynamic linked library in the WINDOWS environment and shared libraries in the UNIX environment and the distributed resource management system to enhance the program capabilities for solving a simulation optimization problem.

5. Development of a Master-Worker server via a file system that solve the simulation optimization.

6. Development of an alternative nonlinear programming solver that uses the mixed complementarity solver.

7. Assembly of the mixed complementarity solver, automatic differentiation and merit function to solve a nonlinear program.

The remainder of this thesis is organized as follows. In chapter 2, we describe the quadratic simulation optimization and its implementation. In chapter 3, we detail the parallel implementation of the quadratic simulation optimization using CONDOR and a Master-Worker server. In chapter 4, we describe the nonlinear programming solver using the mixed complementarity engine and automatic differentiation. Chapter 5 gives conclusions and outlines future research for both simulation optimization and the alternative nonlinear programming solver.

Chapter 2

Simulation optimization using quadratic models

2.1 Simulation optimization model

The aim of our work is to use pre-existing algorithms available via a modeling language to determine an optimal solution of a sample path simulation model. We regard the simulation function, $S_r : \mathbb{R}^n \to \mathbb{R}^m$ as a mapping from a decision variable, $x \in \mathbb{R}^n$, to a simulation response, $y \in \mathbb{R}^m$. The goal-driven simulation problem can be cast as

$$\begin{array}{rl} \min & f(x,y) \\ \text{s.t.} & y = S_r(x) \\ & (x,y) \in B \end{array} \tag{1}$$

where $f : \mathbb{R}^{n+m} \to \mathbb{R}$ is an objective function, y is a variable that holds the simulation output and B specifies additional constraints imposed on the variables. As a simple motivating example, consider an M/M/1 queue with exponentially distributed inter-arrival and service times. We might want to improve the service rate by providing additional training to a server, for example an employee, resulting in a decrease in the amount of time customers have to spend waiting in the queue. An optimization problem might be to determine the amount of training that minimizes the total cost; that is, the cost for training and an additional penalty for lost customer goodwill. For concreteness, assume the penalty is a linear cost of the waiting time, resulting in the following model:

$$\begin{aligned} \min \quad & (\mu - c)^2 + w \\ \text{s.t.} \quad & w = S_r(\mu) \\ & \lambda \leq \mu \end{aligned} \tag{2}$$

where λ is an inter-arrival rate, w denotes an average waiting time of a customer, μ is a service rate variable, c is a constant factor and S_r is an external equation that maps to the external simulation module.

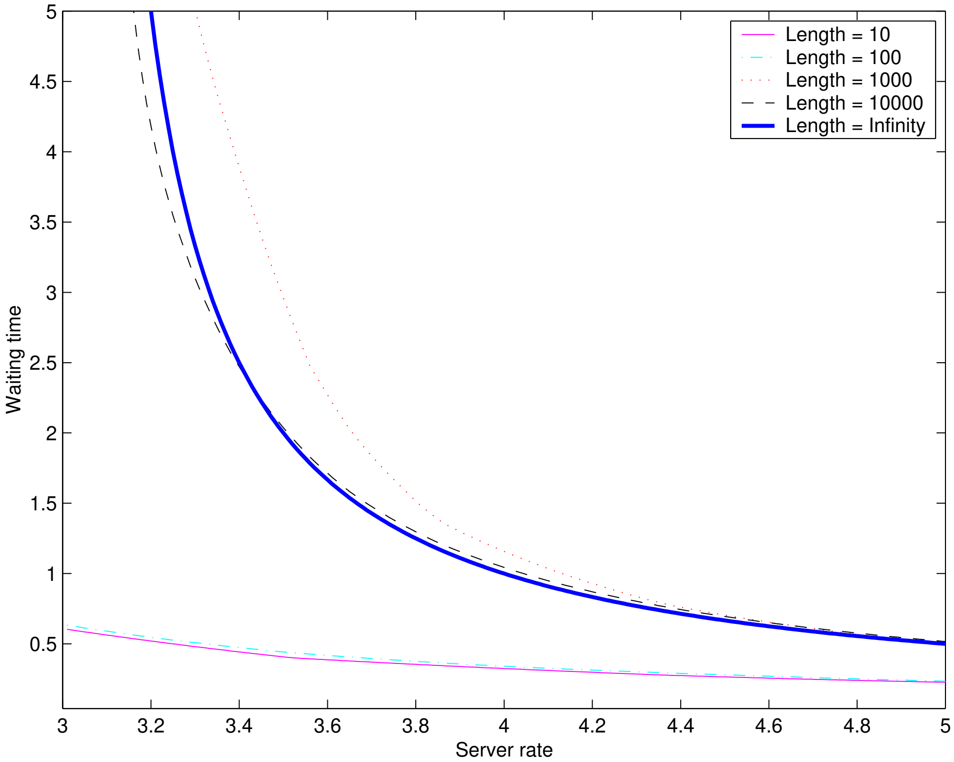

Figure 5: Plot of the simulation function (waiting time) for the fixed random seed.

Figure 5 shows five simulation functions, $S_{10}, S_{100}, S_{1000}, S_{10000}$ and S_∞ for a fixed random seed. As the simulation length increases, the simulation function becomes close to the steady-state simulation function. Therefore, in this case, the minimizer of the long-run simulation function is close to the minimizer of the steady-state simulation function.

Analytically, the average waiting time of the S_∞ can be computed as $w = \frac{1}{\mu-\lambda}$ for an inter-arrival rate λ. We fix the inter-arrival rate at 3. Thus, our M/M/1 simulation optimization approaching the steady state is equivalent to the problem

$$\begin{aligned} \min \quad & (\mu-4)^2 + w \\ \text{s.t.} \quad & w = \tfrac{1}{\mu-3} \\ & \mu \geq 3 \end{aligned} \tag{3}$$

The optimal solution is at $\mu = 4.297$ with an objective value of 0.859.

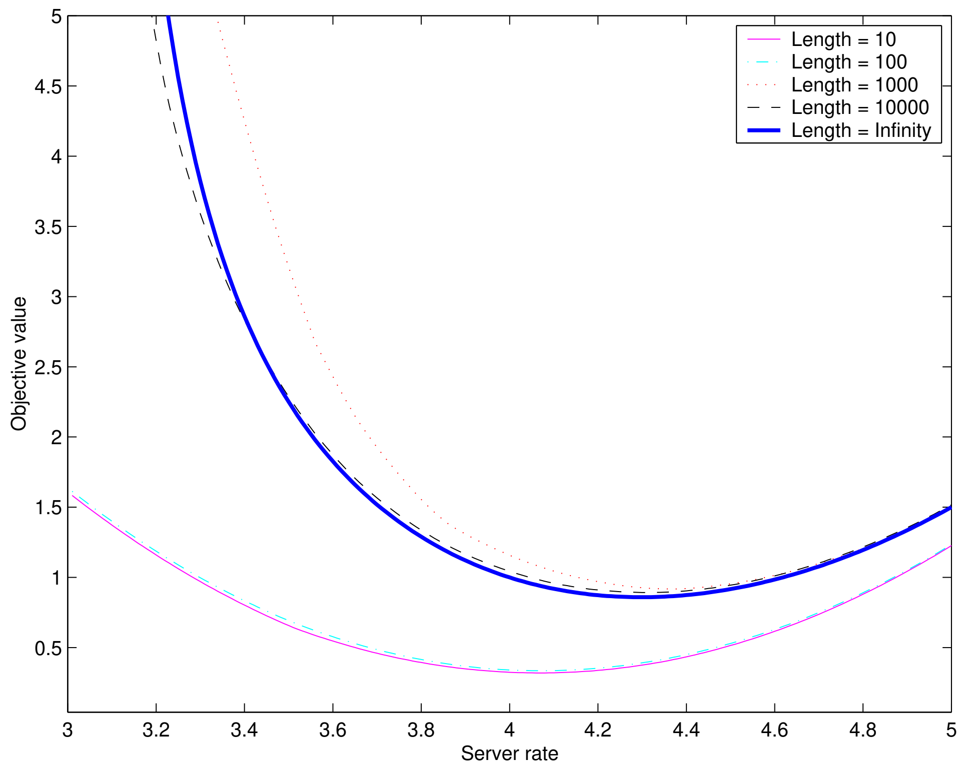

Figure 6: Plot of the objective function of M/M/1 model for the fixed random seed.

Figure 6 plots the objective function for the same sequence of simulation functions. For S_{10000}, the optimum value of this function is very close to the minimizer of S_∞.

For most practical simulation problems, S_r is not available analytically; the simulation function is provided as an oracle that is given the decision variable's choices and produces the outputs or simulation results. To include the non-algebraic function within a modeling language system, we have to use an external equation that links to the simulation module as described in section 2.2. Furthermore, standard nonlinear programming algorithms require an estimate of the derivative of S_r. We show how to achieve this in section 2.3.

2.2 External equations in GAMS

A modeling language system is supposed to facilitate the communication between a practitioner, who wants an answer to an optimization problem, and a mathematical researcher who develops and implements an optimization algorithm. It provides a high-level language that represents optimization problems (such as a linear program, an integer program, a nonlinear program or a mixed complementarity program) and manipulates the results of the solution. A practitioner models a problem using algebraic formulae without worrying about the mechanics of finding the solution. Conversely, a mathematical programmer concentrates on applying a mathematical technique to a standard problem interface that is supplied by the modeling language without worrying about collecting data, writing constraints, or producing a report.

Generally, a model in a modeling language is comprised of four components; data, variables, equations (both equalities and inequalities) and modeling commands. The first component, data, defines a particular instance of the optimization problem that is given by a practitioner, for example, the arrival rate of the M/M/1 queue. Data is also used to define the domain or the set of variables in the problem. For an M/M/c queue, a set of decision variables may be identified as $1, ..., c$ for each server. Therefore, each service rate variable is referred to as $s(1), ..., s(c)$. The second component, variables or decision variables, define the changeable quantities of the optimization problem that a mathematical solver can vary in order to reach the solution. For example, the M/M/1 model in (2) has a service rate as a variable. The third component, equations or constraints, defines the relationship between variables and data that are derived from the problem instance such as bound constraints, budgetary constraints, etc. The last component, modeling commands, is different from the other three components. It is not used to describe the optimization model itself but rather specifies the interaction between a practitioner and the modeling language; examples of these commands are 'model', 'display', 'solve', etc.

Even though a modeling language offers all the benefits of expressing a relationship among variables using mathematical formulae, there is still a limitation on representing some relations, such as a simulation model which requires the use of an external computation outside of the modeling language system. The simulation module is often implemented using a traditional programming language such as C, FORTRAN, C++, Java, etc and is very complex. In practice, it would often be impossible to rewrite the simulation code in the modeling language. In addition, source codes of several simulations may not be available because they have been lost after years of use.

We are interested in using the GAMS modeling language, which extends its applicability by communicating with an external module using a dynamic linked library as in the WINDOWS environment or a shared library as in the UNIX environment. Internally, GAMS defines an external equation as a special mapping that takes an index of the coefficient of the decision variables in the modeling language to the index of the variables in the external module. It also maps the integer value in the equation right-hand-side of the modeling language to an index of the simulation output.

Typically, GAMS has no control over the return values from a simulation module which can be negative, positive or zero depending on the particular simulation run.

GAMS suggests incorporating an external function by using a binding variable and an equation. A binding variable is a regular GAMS variable that will take on the same value as the external function (simulation output) when the equation is satisfied at level zero. The value of this binding variable is automatically set during the optimization process and should not be set by the external module. Thus, a practitioner is required to write an external module that returns the difference of the simulation output and the corresponding binding variable. When the external equation is satisfied, the corresponding binding variable takes on the value of the external function at the given input.

Consider an external function that takes two inputs $x(1)$ and $x(2)$ and provides an output $f(1)$.

$$\text{exteqn.. } 1 * mu + 2 * w \text{ =x= } 1; \tag{4}$$

In GAMS, the external equation (4) shows an input variable mu and a binding variable w that passed to the external module as the first and second variables. The number 1 on the right-hand-side of `=x=` indicates the index in the output vector from the external module. The left hand side declares that $x(1)$ corresponding to mu and $x(2)$ corresponding to w, respectively. The external module returns $f(1) - x(2)$ as its output which is forced to zero by the nonlinear solver resulting in $x(2) = f(1)$. Therefore, $x(2)$ will hold the output simulation of the M/M/1 simulation when $mu = x(1)$.
The general GAMS syntax of an external function is

$$\text{exteqn.. } 1 * var_1 + 2 * var_2 + ... + k * var_k \text{ =x= } m; \tag{5}$$

where m is the m^{th} index of the output vector and k is the total number of variables in the simulation module. The integer coefficient of each variable i identifies a mapping between the GAMS internal variable, var_i, with the i^{th} decision variable of the external module. Note that m of these decision variables are binding to the m outputs of the simulation value. If n is the total number of decision variables and m is the dimension of a simulation output, then $k = n + m$.

FORTRAN

```
      Integer Function GEFUNC (icntr, x, f, d, msgcb)
C Control Buffer:
      Integer icntr(*)
C Numerical Input and Output:
      Double Precision x(*), f, d(*)
C Message Callback Routine:
      External msgcb
```

C

```
GE_API int GE_CALLCONV
gefunc (int *icntr, double *x, double *f, double *d, msgcb_t msgcb)
```

Figure 7: Interfaces for external function call in GAMS.

There are two available interfaces for communicating with an external equation in GAMS; FORTRAN and C. The interfaces are used to pass values of variables, a simulation function and a gradient evaluation between GAMS and the external module. Figure 7 shows both interfaces in FORTRAN and C programming language where `x` is a vector of decision variables including any binding variables, `f` is an output vector, `d` is a derivative evaluation at `x` and `msgcg` is a buffer in which external module reports a message back to GAMS.

Additionally, GAMS passes a problem description via an integer vector `icntr`, which contains problem information such as the number of external equations, the number of external variables, the operational mode such as initialization mode, termination mode or evaluation mode. The interface function returns 0 if it succeeds, 1 if there is a recoverable error, or 2 if there is a fatal error and the program should not continue.

During the initialization mode, `icntr[I_Mode] = 1`, an external module allocates memory, reads initial data from a file or a database, and initializes all program variables. For the termination mode, `icntr[I_Mode]=2`, an external module deallocates memory, reports statistics, closes files, or does other clean-up tasks.

The major mode is the evaluation mode, `icntr[I_Mode] = 3`. In this mode, the external module computes a function if the `icntr[I_Dofunc]` is set to 1 and computes a derivative if the `icntr[I_Dodrv]` is set to 1. The function value of the external module must return the difference between the binding variable and the simulation output. Therefore, it returns zero when the binding variable contains the output of the simulation. For complete details of the GAMS external function interface see the GAMS documentation available at `http://www.gams.com/docs/extfunc.htm`.

Consider the following nonlinear program example,

$$\begin{aligned} \min \quad & x + y \\ \text{s.t.} \quad & x^2 + y^2 \geq 1 \\ & (x, y) \in I\!R^2 \end{aligned} \tag{6}$$

```
Variables obj, x, y;

Equations objective 'The objective function',
          circle    'The circular relation between x and y';

objective..  obj =e= x + y;
circle..     sqr(x) + sqr(y) =g= 1;

model plane /objective,circle/;
solve plane using nlp minimizing obj;
```

Figure 8: Regular GAMS model for (6) .

This can be written as the nonlinear program in GAMS as in figure 8. Figure 9 illustrates the use of the external function. In this formulation, the `circle` equation identifies the original circular relationship between `x` and `y` which states that the point `(x,y)` must lie on or outside of the circle of radius 1. An additional binding variable `r2` is added to capture the circular relation that $x^2 + y^2 = r2$ as in `excircle`.

```
Variables obj, x, y, r2;

Equations objective 'The objective function',
          excircle  'The external circular relation';
          circle    'Radius equation';

objective..  obj =e= x + y;
excircle..   1*x + 2*y + 3*r2 =x= 1;
circle..     r2 =g= 1;

model plane /objective,circle/;
option nlp=conopt2;
solve plane using nlp minimizing obj;
```

Figure 9: GAMS model for the model (6) using the external equation feature..

We also have to implement the external module that corresponds to the external equation in GAMS.

```
      include 'geheader.inc'
      include 'geindex.inc'
      Integer Function GEFUNC(Icntr, X, F, D, MSGCB)
      Implicit none
      Integer Icntr(*), i, neq, nvar, nz
      double precision X(*), F, D(*), realF
      SAVE    neq, nvar, nz
      GEFUNC = 0
      if(icntr(I_Mode) .eq. 1) then
         neq  = 1
         nvar = 3
         nz   = neq * nvar
      elseif(Icntr(I_Mode) .eq. 2) then
      elseif (Icntr(I_Mode) .eq. 3) then
         if(Icntr(I_Eqno) .le. neq) then
C           Function and Derivative Evaluation Mode
            if(Icntr(I_Dofunc) .ne. 0) then
C              Compute function value
               realF = X(1)*X(1) + X(2)*X(2)
               F = realF - X(3)
            endif
            if(Icntr(I_Dodrv) .ne. 0) then
C              Compute derivative value
               D(1) = 2*X(1)
               D(2) = 2*X(2)
               D(3) = -1
            endif
         else
         Call GEstat(Icntr,'**** Unexpected index of F')
         GEFUNC = 2
         endif
      else
         Call GEstat(Icntr,'**** Unexpected mode value')
         GEFUNC = 2
      endif
      return
End
```

The FORTRAN code given above computes a value of $x^2 + y^2$ and puts it in the `realF` variable. However this function returns the value of `realF - X(3)` where `X(3)` is a binding variable that corresponds to `r2`. For the derivative computation, the program computes and returns $2x$ and $2y$ for `X(1)` and `X(2)`, respectively. It also returns -1 for the derivative of the binding variable `X(3)` because it appears in the external module as `realF - X(3)`.

This FORTRAN program needs to be compiled as a dynamic linked library in the WINDOWS NT environment ('plane.dll') and as a shared library in the UNIX environment ('plane.so') and put it in the same directory as the GAMS model. The model can then be run in the normal fashion.

2.3 Gradient approximation using the quadratic model

Standard nonlinear programming software typically requires that a user provide at least first order derivatives for each function appearing in the model (1). GAMS uses automatic differentiation to compute a derivative of all algebraic equations written in GAMS. However, the simulation function S is not defined analytically and only available as an oracle. We therefore construct a meaningful derivative using only function evaluations. Instead of using finite differences, we advocate fitting a low order polynomial to observed simulation output. The reasons for this choice were outlined in Chapter 2.

When the solver requests a function evaluation of an external equation, control is passed to the external module with the appropriate arguments. If evaluation at this point has never been requested before, then the simulation run is made and the result is stored. It then returns the simulation value to the solver. For a derivative evaluation, the point is checked against the previously computed points. If there are enough points to fit the quadratic model, the program applies a least squares fitting method to determine the best quadratic model. For example, our implementation evaluates the simulation at a number of random points chosen in a neighborhood of $x_0 \in \mathbb{R}^n$, and then fits a quadratic model, $A(x_0) := x_0^T Q x_0 + c^T x_0 + d$, in the least squares sense, to the observed simulation function values. Since the quadratic model is symmetric, the program determines only $\frac{n(n+1)}{2} + n + 1$ sampling points. The derivative is passed back to the nonlinear solver as $\frac{dA}{dx}(x_0) := 2x_0^T Q + c^T$. Clearly, the appropriateness of the model depends on the error in the simulation and the size of the neighborhood. We use a statistical test to check the validity of the model allowing for white noise error. If the approximation is deemed poor, we reduce the size of the neighborhood and construct a new model.

Note that while the evaluation of the simulation function $S(x_0)$ may be noisy, we always return this as the function value of the simulation, rather than the value of the quadratic model at x_0. The principal reason is that regardless of the path of the algorithm, this value will always be the same by the deterministic assumption of the sample path approach. However, if we were to choose different random points in the neighborhood of x_0, the quadratic model could change. Thus, $A(x_0)$ might have a different value depending upon this choice of points.

One problem with this approach is that, by assumption, the simulation evaluations are expensive. Therefore, the code collects (and stores) all previously computed values for points in the neighborhood of x_0 to reduce the number of simulation evaluations performed. If the total number of points is not enough to fit the quadratic function, then an appropriate number of uniformly distributed random points in the neighborhood of x_0 within a radius r are generated. The simulation module is called to compute each of the function values at the newly generated points.

2.3.1 Least Squares approximation

Once all of the simulation evaluations have been collected, we fit the quadratic model in a least squares sense. Let S be the simulation function from $\mathbb{R}^n \rightarrow \mathbb{R}^m$ and S_l be the l^{th} component of the simulation function. The quadratic approximation of S for each component $l = 1, \ldots, m$ is

$$S_l(x) \approx A_l(x) := x^T Q^l x + {c^l}^T x + d^l.$$

Let $x^1, x^2, x^3, \ldots, x^{np}$ be the sampling points used to approximate S_l, where np is greater than the number of unknown coefficients. The least squares problem we solve is

$$\min_{Q,c,d} \sum_{k=1}^{np} \left((x^k)^T Q x^k + c^T x^k + d - S_l(x^k) \right)^2,$$

The coefficients, Q, c and d, are the variables of the problem and that x^k is given data. Since Q is symmetric, only the upper triangular elements of Q are needed. We then have $np \geq \frac{n(n+1)}{2} + n + 1$. Therefore, we minimize

$$\sum_{k=1}^{np} \left(\sum_{i=1}^{n} \left(Q_{i,i}(x_i^k)^2 + 2 \sum_{j>i}^{n} Q_{i,j} x_i^k x_j^k + c_i x_i^k \right) + d - b_k \right)^2,$$

where $b_k = S_l(x^k)$ for $k = 1, \ldots, np$.

Let $z = (Q_{11}, Q_{12}, \ldots, Q_{nn}, c_1, \ldots, c_n, d)$ with $p = \frac{n(n+1)}{2} + n + 1$ and define C^T by

$$\begin{bmatrix} x_1^1 x_1^1 & x_1^2 x_1^2 & \ldots & x_1^{np} x_1^{np} \\ 2x_1^1 x_2^1 & 2x_1^2 x_2^2 & \ldots & 2x_1^{np} x_2^{np} \\ \vdots & \vdots & & \vdots \\ 2x_1^1 x_n^1 & 2x_1^2 x_n^2 & \ldots & 2x_1^{np} x_n^{np} \\ x_2^1 x_2^1 & x_2^2 x_2^2 & \ldots & x_2^{np} x_2^{np} \\ \vdots & \vdots & & \vdots \\ 2x_2^1 x_n^1 & 2x_2^2 x_n^2 & \ldots & 2x_2^{np} x_n^{np} \\ \vdots & \vdots & & \vdots \\ x_n^1 x_n^1 & x_n^2 x_n^2 & \ldots & x_n^{np} x_n^{np} \\ x_1^1 & x_1^2 & \ldots & x_1^{np} \\ \vdots & \vdots & & \vdots \\ x_n^1 & x_n^2 & \ldots & x_n^{np} \\ 1 & 1 & \ldots & 1 \end{bmatrix}$$

The optimality conditions of the least squares problem are the normal equations

$$C^T C z = C^T b. \tag{7}$$

We form the matrix $C^T C$ explicitly, and use the LAPACK library [3] to factor and solve this symmetric (positive-definite) system of linear equations. When $m > 1$, the system is repeatedly solved, once for each different value of b.

2.3.2 Statistical Tests

We determine the suitability of our quadratic model using a statistical test. The first test is the coefficient of determination, R^2 [1], that determines how well the quadratic model predicts values for the simulation function. The second test is the the Cramér-von

Mises statistic, W^2 [81, 82], that determines the goodness-of-fit of the error distribution to some known empirical distribution functions such as normal distribution. The last test is the coefficient of skewness [1] that determines the lack of the symmetry of the error distribution in order to identify a group of extreme values.

Coefficient of determination

After the coefficients of the quadratic model have been determined, the model can be used to predict the simulation value within that neighborhood. From a statistical point of view, this is considered as a linear regression on the coefficients of the quadratic model to the simulation data.

Denote b_k as the simulation value at x^k and $\bar{b}$ as a sample mean of b_k's. The coefficient of determination, R^2, is the ratio of the regression variation over the total variation that determines the predictive power of the quadratic model. It is computed as

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

where SSR is the regression sum of squares defined as

$$SSR = \sum_{k=1}^{np} \left((x^k)^T Q x^k + c^T x^k + d - \bar{b}\right)^2$$

and SST is the total sum of squares defined as

$$SST = \sum_{k=1}^{np} (b_k - \bar{b})^2$$

and SSE is the error sum of squares defined as

$$SSE = \sum_{k=1}^{np} \left((x^k)^T Q x^k + c^T x^k + d - b_k\right)^2.$$

It is clear that R^2 lies between 0 and 1 because SSR and SST are positive and SSR is less than SST. If the error sum of squares is close to zero or R^2 is close to one, then the quadratic model can be used to predict the simulation values. A gradient evaluation from the accepted quadratic model using R^2 will be very close to other gradient approximation methods such as the finite differences or infinitesimal perturbation analysis method.

Even when R^2 is small, we would like to accept the quadratic model if the model exhibits white noise error distribution. In the next section, we show how to use the W^2 statistic to accept the quadratic model based on the goodness-of-fit test. This shows a distinction of our gradient computation from the others.

Cramér-von Mises Statistic

The Cramér-von Mises Statistic was developed by Cramér in his book [15] to test for a distribution of data with the continuous and completely specified distribution such as the normal distribution. We apply this test under assumption that the quadratic model returns an appropriate derivative if the error has a normal distribution. Given $e_1, \ldots, e_n$ from a continuous population distribution $G(\cdot)$, let $F_n(\cdot)$ be the empirical distribution function of e_i. The null hypothesis test is

$$H_0 : G(e) = F(e; \theta)$$

where $F(\cdot; \theta)$ is a given distribution (typically normal or exponential) with parameter θ. The Cramér-von Mises statistic is

$$W^2 = n \int_{-\infty}^{+\infty} \{F_n(e) - F(e; \theta)\}^2 dF(e; \theta).$$

For the normal distribution with unknown mean and variance, we have $F(\cdot; \theta) = N(\cdot; \bar{e}, s_e)$ where $\bar{e}$ is the sample mean and s_e is the sample standard deviation. We test the distribution of the difference between the simulation run and the quadratic model to determine if the error has a normal distribution.

During the testing step we perform the following:

1. Sort the data (errors between the simulation and quadratic prediction model) as $e_1 \leq e_2 \leq \ldots \leq e_n$.

2. Compute the sample mean $\bar{e}$ and sample standard deviation s_e.

3. Calculate $w_i = (e_i - \bar{e})/s_e$.

4. Compute $z_i = CDF(w_i)$ where $CDF(\cdot)$ is the cumulative probability of a standard normal distribution.

5. Calculate the W^2 statistic from these (sorted) z-values where

$$W^2 = \sum_{i=1}^{n}(z_i - \frac{2i-1}{2n})^2 + \frac{1}{12n}$$

6. Update W^2 to reflect the assumption of unknown mean and unknown variance, $W^2 = W^2(1 + \frac{0.5}{n})$.

7. Compare this value against the T^* statistical table (see [81]). We use $T^*_{0.15} = 0.091$, $T^*_{0.10} = 0.104$, $T^*_{0.05} = 0.126$, $T^*_{0.025} = 0.148$, and $T^*_{0.01} = 0.178$.

If W^2 is larger than T^*_α, then we reject the null hypothesis H_0 at the significance level α.

In our procedure, we fix the significance level $\alpha = 0.05$, then compute W^2 and compare it with $T^*_{0.05}$. If the null hypothesis is accepted, the error is due to white noise, the trend in the approximation is deemed to coincide with the actual trend of the simulation function. The quadratic approximation is then used.

If we reject this null hypothesis, i.e. $W^2 > T^*_{0.05}$, then we re-fit the quadratic model by removing the extreme values using a smaller radius that is determined by the coefficient of skewness.

Coefficient of Skewness

The coefficient of skewness [1] is a measure of the lack of symmetry in data. If data is distributed symmetrically from the left and the right of the center point, then the skewness is zero. For example, the normal distribution and uniform distribution have skewness equal to zero. When the skewness is negative, the distribution of data is more weighted to larger values than smaller values. For positive skewness, data with smaller values are more prominent than data with larger values. The Chi-square distribution is an example of positive skewness.

The skewness sk is computed from $x_0, \dots, x_n$ by

$$sk = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{(n-1) \times s_e^3}$$

where s_e is the sample standard deviation and $\bar{x}$ is the mean of $x_0, \dots, x_n$.

We use the skewness to identify outliers or extreme values and partition data into small groups according to their skewness value. We then eliminate an undesirable group from our model building by determining a new reduced radius. If the skewness is inside the range of [-0.5, 0.5], then the algorithm updates the radius using a predefined value. If the skewness is less than -0.5, most of the simulation values have larger values. The algorithm aims to discard the smallest block of data values which contains extreme values. The new radius is computed as the largest radius from x_0 to other points excluding points in the removed block. The algorithm performs similarly for the skewness that is greater than 0.5.

In the case that x_0 is one of the points in the extreme block, the new radius is determined by the largest radius from x_0 to all points in the block because we want to locally fit the quadratic model to this x_0. This procedure is repeated at most `MAX_I` number of times. If no quadratic model has been accepted then the algorithm returns the derivative of the recent quadratic model and marks this point x_0 as having a poor derivative approximation. The next time, the derivative information is request at this point, then the algorithm will start using the smaller radius to build the quadratic model.

2.4 Detailed implementation of QSO solver

The quadratic simulation optimization solver, called QSO, uses the existing nonlinear optimization solver, CONOPT, to find its solution point. The function call is returned using a simulation run to the QSO algorithm, while the derivative call is returned as the derivative of the quadratic model built from surrounding points and evaluation requests from the simulation module as necessary.

2.4.1 Function computation of QSO

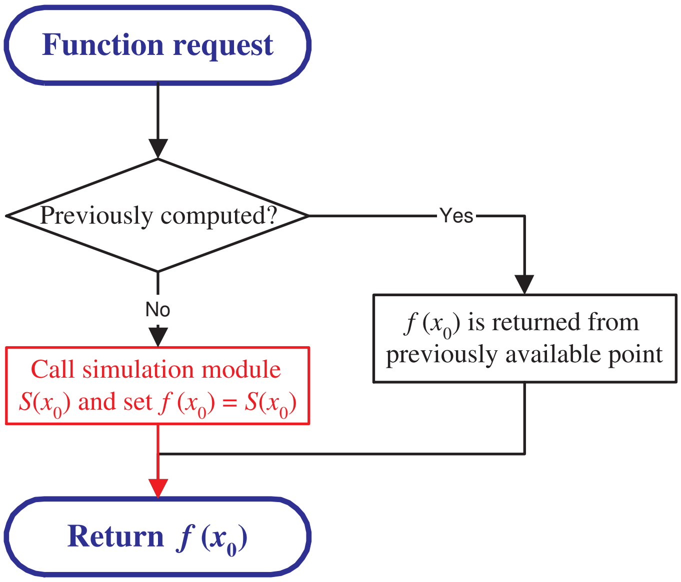

Figure 10: A flow chart for computing simulation function at a point x_0.

Figure 10 shows the computation of the simulation function at point x_0. The algorithm starts by comparing the current evaluation point with all stored points. If the point is not close to the previously computed points, then it makes a system call to the simulation module of the form

```
sim_run input_file output_file
```

where `sim_run` is the binary executable simulation program, `input_file` contains the current point for function evaluation. After the execution is completed, the `output_file` contains the output simulation values. The QSO program then reads the simulation result from the `output_file` and returns it to GAMS. To reduce the number of simulation calls, the algorithm keeps track of all previous computed simulation points in a global array which will be rotated when the array is full. Nevertheless, the call to a function is guaranteed to return the same result either from this global array or making the simulation run with the same random number sequence.

2.4.2 Derivative computation of QSO

A detailed flow chart of the complete implementation of a derivative evaluation is given in figure 11. This chart summarizes the information contained in section 2.3.

The algorithm collects all previously computed points surrounding x_0 using the starting radius, r. If the number of collected points is not sufficient to fit the quadratic model, then it uniformly generates random points within the radius r. To compute all these new points, the simulation module is called. If the quadratic model is rejected due to R^2, then it reduces the radius to avoid any extreme points or outliers. It uses the coefficient of skewness to determine the block of data to ignore for the next quadratic approximation using points in this neighborhood of x_0. If the current quadratic function does not exhibit an appropriate fit, then it removes outliers or extreme points using the skew index and updating the radius. In the case that x_0 is among the extreme points, the algorithm uses the current quadratic function to compute the derivative at x_0 and return

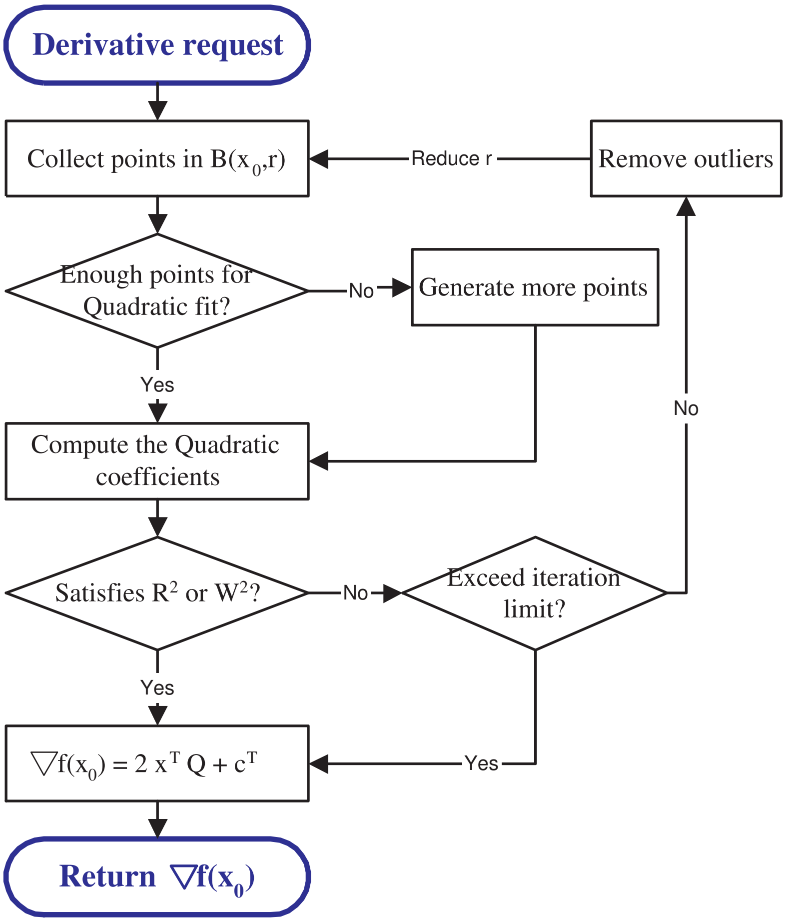

Figure 11: A flow chart for computing derivative at a point x_0.

it. In addition, the point is marked as an unfavorable differentiable value, so it will try to find a better quadratic model the next time that CONOPT requests a derivative at this point. If the quadratic function exhibits a good fit, the algorithm records the radius and returns the derivative by computing $2Qx_0 + c^T$. We also limit the number of iterations in this routine. If we exceed the iteration limit, then we use the last quadratic estimation to compute the derivative at x_0.

2.5 Examples and results

The semantics for using external functions in GAMS dictate that the user writes a function and compiles it in such a manner that the code can be linked dynamically with the solver executable. We have implemented the quadratic approximation code as outlined in this chapter for this purpose. The remaining piece is the simulation routine which is incorporated into the quadratic approximation code. As the syntax for calling the simulation routine varies, we only require that a user writes a small interfacing function that calls their simulation for a given input and returns the outputs. The function can simply call the simulation if it is available in C or FORTRAN, or it can use system calls to run it as an external program, as we outlined previously.

We have written such routines for three different simulations. We have incorporated these simulations into optimization problems that are formulated within GAMS. The remainder of this chapter details the simulation optimizations and the results obtained on them.

2.5.1 M/M/1 Queue

The first problem optimizes a stable M/M/1 queue from (2) to minimize average waiting time. This problem can be solved analytically, providing us with a mechanism to check the validity of our optimization approach. Analytically, the average waiting time is $w = \frac{1}{\mu - \lambda}$ for an inter-arrival rate λ. For our testing, we fix the inter-arrival rate at 3. Thus, our M/M/1 simulation optimization approaching the steady state is equivalent to the problem

$$\begin{aligned} \min \quad & (\mu - 4)^2 + w \\ \text{s.t.} \quad & w = \tfrac{1}{\mu - 3} \\ & \mu \geq 3 \end{aligned} \tag{8}$$

The optimal solution is at $\mu = 4.297$ with an objective value of 0.859.

To test our optimization approach, we used simulations with 10000, 100000, and 1000000 customers. The first 1% of customers were ignored to avoid initial bias. Tables 1 and 2 show details of the output from the M/M/1 simulation optimization problem based on different numbers of sampling points. For all runs, a starting value of $\mu = 3.0$ was used with an initial radius of 1.0 for the quadratic model neighborhood and $R^2 = 0.99999$. We ran these results on a Pentium III 600 MHz machine running WINNT.

Table 1: Comparison of M/M/1 optimization parameterized by length of simulation run and number of points sampled in quadratic model without the W^2 statistic

Simulation length	*Sampling pts. (np)*	*Runs (sim.)*	*Time (sec.)*	*Obj. value*
10000	7	195	3	0.9015
	8	194	3	0.9015
	9	214	3	0.9015
	14	211	3	0.9015
	19	213	3	0.9015
	24	411	6	0.9015
100000	7	186	25	0.8536
	8	194	25	0.8536
	9	205	26	0.8536
	14	241	31	0.8536
	19	324	42	0.8536
	24	326	42	0.8536
1000000	7	134	170	0.8586
	8	179	226	0.8586
	9	208	263	0.8587
	14	220	278	0.8586
	19	237	507	0.8586
	24	241	553	0.8586
Infinity				0.8592

For the same simulation length, our algorithm achieves the same optimal solution independent of the number of sampling points. As the length of the simulation increases,

Table 2: Comparison of M/M/1 optimization parameterized by length of simulation run and number of points sampled in quadratic model with the W^2 statistic

Simulation length	*Sampling pts. (np)*	*Runs (sim.)*	*Time (sec.)*	*Obj. value*
10000	7	175	3	0.9015
	8	153	2	0.9015
	9	81	1	0.9016
	14	200	3	0.9015
	19	205	3	0.9015
	24	447	7	0.9015
100000	7	205	35	0.8536
	8	165	41	0.8536
	9	205	54	0.8536
	14	212	53	0.8536
	19	337	66	0.8536
	24	335	81	0.8536
1000000	7	221	514	0.8586
	8	180	405	0.8586
	9	226	524	0.8586
	14	305	702	0.8586
	19	392	900	0.8586
	24	466	1059	0.8586

the sample-path optimization solution obtained by our method converges to the correct solution as predicted by the theory. The overall solution time depends heavily on the length of the simulation run. However, these tables give no indication that the use of the W^2 statistic is beneficial. The simulation runs are long enough that the function values perceived by the optimization code are not noisy and the overall simulation function $S(\mu)$ is smooth and well-behaved. The remainder of the examples use more realistic simulation codes that indicate more benefits of the W^2 statistic.

2.5.2 Telemarketing system example

A more interesting example comes from simulating a telemarketing system where we have a fixed number of operators answering calls. The number of customers on hold (waiting for service) is fixed. If a customer is denied entry into the queue, they are given a busy signal and there is a probability p that they will call back after waiting an exponentially distributed amount of time. Those that do not call back result in a lost sale. We want to choose the service rate on the operators to minimize some weighted sum of operator training costs and lost sales. A schematic overview of the simulation is given in Figure 12.

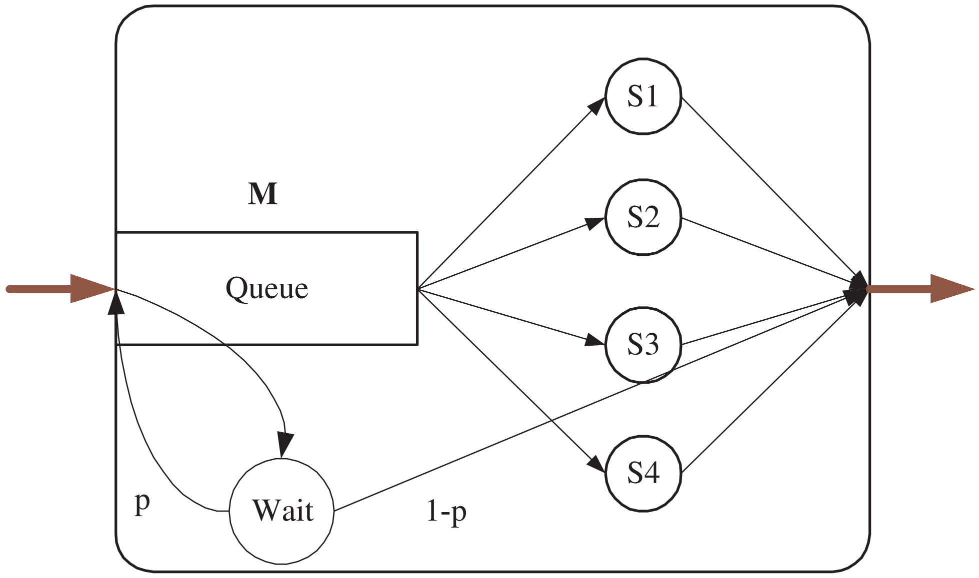

Figure 12: Telemarketing Simulation Structure.

We will assume the system contains 4 operators and a fixed queue size of $M = 100$. Initially, the operators have a service rate of $\mu_i = 0.5$. The inter-arrival times of customers first entering the system is exponentially distributed with an inter-arrival rate of $\lambda = 3$. The probability that a user will call back is $p = 0.1$ with an exponentially distributed waiting time of 0.4. The variables in the optimization are the service rates which are bounded below by 0.5 and the outputs from the simulation are the the percentage of customers lost and the average waiting time.

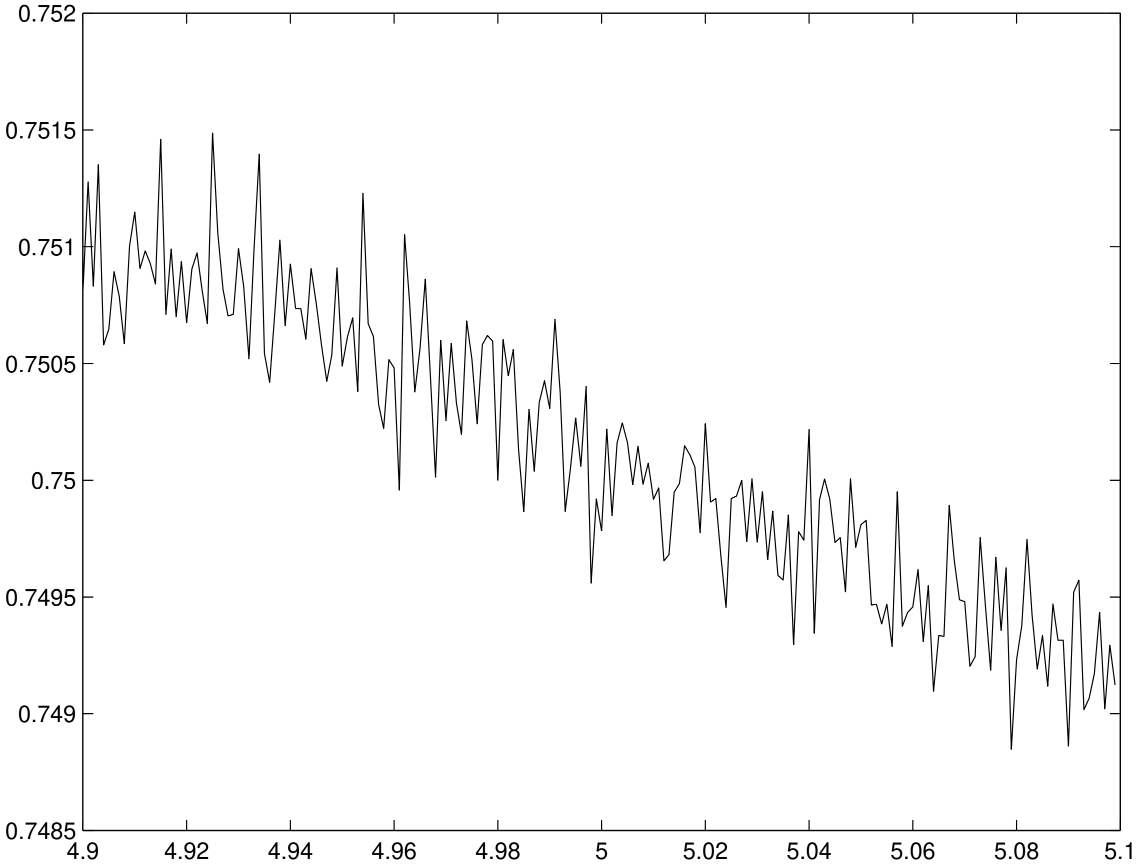

Figure 13: Telemarketing Simulation Structure.

This simulation model is very noisy due to the probability of customers leaving the system without being served. Since the simulation is coded with a single random input stream, this can lead to significant changes in simulation outputs for small variations in the input parameters. Figure 13 shows how the percentage of calls lost change as the service rate for the first operator is increased. Note the output varies dramatically for small variations in inputs, but the overall shape of the function is clear. We expect the W^2 statistic to be beneficial in solving the optimization problem, since it attempts to reduce the effects of noise by generating quadratic models of the overall trend in the simulation functions.

Table 3: Comparison of call-waiting simulation optimization parameterized by length of simulation run and number of points sampled in quadratic model without the W^2 statistic

Simulation length	*Sampling pts. (np)*	*Runs (sim.)*	*Time (sec.)*	*Obj. value*
10000	35	582	54	1.4663
	50	887	45	1.0282
	65	885	71	1.2933
	80	1888	141	1.0277
	95	2144	160	1.0280
	110	1590	117	1.0279
100000	35	999	727	1.1303
	50	744	536	1.4641
	65	746	536	1.3358
	80	1013	722	1.1172
	95	1679	1181	1.1187
	110	1673	1201	1.1249
1000000	35	343	2993	1.5324
	50	533	4475	1.5331
	65	785	5009	1.1783
	80	501	2945	1.1707
	95	735	3940	1.1809
	110	1279	6337	1.2117

The goal for optimizing this call-waiting simulation is to achieve the minimum number of customers lost due to the busy signal of the servers. Since the servers behave identically in the system, there are many solutions that satisfy our goal. To specify a reasonable optimization problem, we define an objective function as

$$obj = \sum_{i=1}^{4} w_i(\mu_i - l) + 100 \times lost$$

where l is the lower bound for all service rates, $lost$ is the percentage lost of customers in the system, μ_i is the service rates of i^{th} server and w_i is the weight on the i^{th} server.

Different weights on each server correspond to different costs for training. For the runs presented $w_1 = 1$, $w_2 = 5$, $w_3 = 10$ and $w_4 = 15$, respectively.

Table 4: Comparison of call-waiting simulation optimization parameterized by length of simulation run and number of points sampled in quadratic model with the W^2 statistic

Simulation length	*Sampling pts. (np)*	*Runs (sim.)*	*Time (sec.)*	*Obj. value*
10000	35	556	45	1.0281
	50	760	58	1.0280
	65	1224	93	1.0278
	80	1775	134	1.0277
	95	2517	192	1.0276
	110	1773	138	1.0278
100000	35	958	698	1.1191
	50	978	705	1.1186
	65	993	712	1.1354
	80	1013	722	1.1431
	95	1049	745	1.1214
	110	2127	1524	1.1216
1000000	35	881	7168	1.1760
	50	1696	12908	1.1655
	65	848	4698	1.2480
	80	1511	10478	1.1666
	95	1031	5580	1.1636
	110	2261	13293	1.2301

Tables 3 and 4 show the details of our results based on different numbers of sampling points. Again the benefit of using additional sampling points to fit the quadratic model is unclear - the results with small values of np are similar to those with large values (except the smaller values execute more quickly). It appears that the results using the W^2 statistic are significantly more robust than those without. This robustness comes at some cost in terms of computing and time. While the precise solution is unknown, the optimization of the longer length simulation appears to give more accurate values for the objective value under independent simulation runs of even greater length.

2.5.3 Tandem Production Line

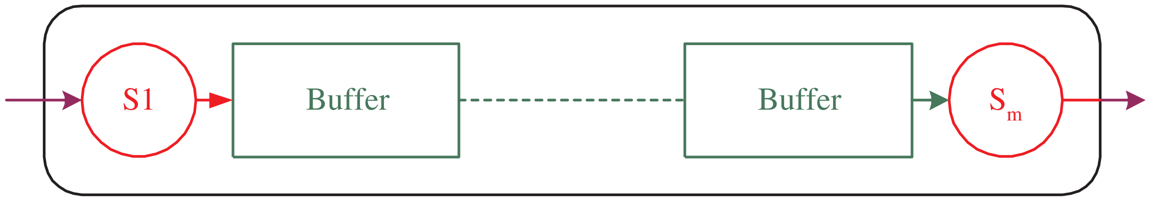

Figure 14: Tandem Production Line Simulation Structure

The final simulation we attempted to optimize in this chapter is a tandem production line composed of m machines and $m-1$ buffers which hold the excess product between machines arranged in series. The product arrives from an external source to the first machine and is then processed by each machine. Progress is blocked when the number of products in a buffer equals the maximum buffer size. The machine then waits until there is an available slot in the buffer. There is also a probability that a machine may fail at an exponentially distributed time. The time to repair a failed machine is also exponentially distributed. The input parameters to the simulation are the machine processing rates, the probability of machine failure, and the rate of repair for each failed machine. The output parameter is the reciprocal of throughput where the throughput is the average processing rate for the entire line. Figure 14 gives a diagram of the tandem production system with m machines.

The actual simulation code that we use was provided by Erica L. Plambeck and is based on tandem production problems from [60]. We use this paper as a basis for comparison here, and hence fix the probability of machine failure and the rate of repair to the values given in that paper. The paper contains 7 cases each with two different starting points for a total of 14 problems. There are two machines in problems 1 and 2, four machines in problem 3, six machines in problems 4 and 5, five machines in problem 6 and fifteen machines in problem 7. Two methods were used to obtain the results reported in the paper. The first method is Bundle-based stochastic optimization (BSO) which is applicable to all of the problems. The second method, single run optimization (SRO), only applies to cases 1 through 5. We obtained the simulation code from the author and used it with our optimization methodology. We use the label QSO to indicate our method.

Table 5: Objective value comparisons between the SRO, BSO, and QSO methods

Case	*Var.*	*SRO*	*BSO*	*QSO*	
				without W^2	*with* W^2
1a	2	7.6899	7.6895	7.6899	7.6899
1b	2	7.7010	7.7008	7.7008	7.7008
2a	2	0.9638	0.9638	0.9637	0.9637
2b	2	1.0070	1.0070	1.0070	1.0070
3a	4	0.7404	0.7404	0.7404	0.7404
3b	4	0.7358	0.7404	0.7356	0.7357
4a	6	0.3956	0.3957	0.3955	0.3955
4b	6	0.3960	0.3960	0.3960	0.3960
5a	6	0.3485	0.3482	0.3465	0.3465
5b	6	0.3450	0.3446	0.3413	0.3413
6a	5		3.3956	3.3950	3.3951
6b	5		3.3977	3.3928	3.3928
7a	15		3.4065	3.4107	3.4107
7b	15		3.4061	3.4043	3.4054

Each simulation run uses 49500 units with 500 units to remove bias, except that 7a and 7b use 90000 units. The starting radius for fitting the quadratic was set to be equal to the total number of machines and R^2 was set to 0.99999. Table 5 compares the optimal solutions found among three methods, BSO, SRO and QSO. We can see that the solutions found by QSO with or without the W^2 statistic are virtually indistinguishable and are all comparable to those found by SRO and BSO. In fact, on problems 5a, 5b, 6a, 6b and 7b, QSO seems to provide the best solutions of all codes. On problem 7a, QSO

Table 6: Tandem production line comparison for the QSO method with and without the W^2 statistic

Case	*Without W^2 statistics*		*With W^2 statistics*	
	Runs (sim.)	*Time (sec.)*	*Runs (sim.)*	*Time (sec.)*
1a	223	87	235	91
1b	240	94	351	140
2a	129	5	130	5
2b	40	2	73	3
3a	570	411	495	361
3b	737	521	666	461
4a	3074	4476	2524	3641
4b	3322	4879	1870	2733
5a	2778	244	2151	184
5b	3481	304	1962	166
6a	1827	106	2430	140
6b	1610	92	906	53
7a	****	50000	****	50000
7b	24210	28468	14443	17636

had more difficulties and we terminated it after it hit a time limit at a slightly worse objective value. By adding an extra constraint that constrains the sum of all the machine rates, we were able to solve 7a to optimality as well. Table 6 shows the total simulation runs and the total time used by our algorithm, with and without the W^2 statistic. These results seem to indicate that for the larger dimension problems the use of the W^2 statistic is preferable.

From Plambeck 1996 [61], she revised the problem and resolve 1a, 1b, 2a, 2b using the inverse of flow rates as the decision variables.

Table 7: Objective value comparisons for the new tandem production line for SRO, BSO, (from the Table 2 page 151 of [61]) and QSO methods

Case	*SRO*	*BSO*	*QSO*	
			without W^2	*with* W^2
1a	0.6896	0.6896	0.6851 (718)	0.6850 (700)
1b	0.6897	0.6896	0.6944 (572)	0.6944 (396)
2a	0.5375	0.5374	0.5363 (806)	0.5363 (280)
2b	0.5374	0.5374	0.5386 (220)	0.5386 (1006)

where the number in the parenthesis represents the number of simulation runs.

The results of SRO and BSO from table 7 are extracted from Plambeck 1996 paper [61]. Our QSO results are found using the same settings which are published in her paper. However, the stream of random numbers are not the same because there are no seeds available in the paper. All solutions from SRO, BSO and QSO are the same at (0.5, 0.5) for case 1 and (0.497, 0.503) for case 2. The total number of simulation budget that she used is 1,000,000 units for both cases. We run our algorithm based on the same simulation length. Our method uses a total simulation budget varying from 22,000,000 to 100,600,000 units. We expect to use more simulations than BSO because we do not modify the simulation source code at all. However, as we now show, we do not need long simulation lengths for QSO to be effective.

From table 8, we apply our algorithm with smaller simulation budgets. As this table shows, the optimal solution from our QSO algorithm with 5,000 simulation length is within 3 significant digits of the optimum solution. If we calculate the number of units required for the results in table 8, we see that significantly fewer units are needed, namely, the total simulation budgets for case 1a, 1b, 2a and 2b are about 3.48, 2.89, 3.64, 3.69

Table 8: Objective value comparisons for the new tandem production line based on different simulation length of the QSO method with and without the W^2 statistic

Length	*Case*	*Without W^2 statistics*			*With W^2 statistics*		
		Objective	*Runs*	*Solution*	*Objective*	*Runs*	*Solution*
1000	1a	0.5761	976	(0.500, 0.500)	0.5761	920	(0.500, 0.500)
	1b	0.6403	342	(0.509, 0.492)	0.6403	1602	(0.509, 0.491)
	2a	0.5283	658	(0.498, 0.502)	0.5283	848	(0.498, 0.502)
	2b	0.5205	1302	(0.501, 0.499)	0.5205	1384	(0.501, 0.499)
2000	1a	0.6440	608	(0.500, 0.500)	0.6440	1068	(0.500, 0.500)
	1b	0.6431	598	(0.500, 0.500)	0.6431	1080	(0.500, 0.500)
	2a	0.5403	924	(0.498, 0.502)	0.5406	826	(0.497, 0.503)
	2b	0.5237	702	(0.498, 0.502)	0.5237	1384	(0.498, 0.502)
3000	1a	0.6769	612	(0.500, 0.500)	0.6769	1310	(0.500, 0.500)
	1b	0.6328	1116	(0.500, 0.500)	0.6328	624	(0.500, 0.500)
	2a	0.5406	638	(0.495, 0.505)	0.5406	1080	(0.495, 0.505)
	2b	0.5352	1074	(0.498, 0.502)	0.5352	700	(0.498, 0.502)
4000	1a	0.6953	692	(0.500, 0.500)	0.6953	892	(0.500, 0.500)
	1b	0.6722	678	(0.500, 0.500)	0.6722	590	(0.500, 0.500)
	2a	0.5381	634	(0.497, 0.503)	0.5381	738	(0.498, 0.502)
	2b	0.5322	1034	(0.497, 0.503)	0.5322	586	(0.497, 0.503)
5000	1a	0.7082	696	(0.500, 0.500)	0.7082	626	(0.500, 0.500)
	1b	0.6944	578	(0.500, 0.500)	0.6944	630	(0.500, 0.500)
	2a	0.5391	728	(0.497, 0.503)	0.5391	734	(0.497, 0.503)
	2b	0.5328	738	(0.497, 0.503)	0.5328	772	(0.497, 0.503)

(million) for QSO without W^2 statistic and 3.13, 3.15, 3.76, 3.86 (million) for QSO with W^2 statistic, respectively. Therefore, we could reduce the total simulation budget of our method by solving the simulation optimization of the initial starting point with small simulation runs and then verifying this optimal solution based on longer simulation length.

Chapter 3

Computation in a distributed environment

3.1 CONDOR and its requirements

An increasing need for large amount of computational power to solve real-life problems causes researchers to search for a robust computational tool that supplies massive computational power with ease of expandability. The use of one centralized supercomputer machine is limited and somewhat obsolete because of the cost of buying and maintaining the machine. CONDOR [54, 21, 53] is a meta-computing environment [34] that is developed to exploit the computational resources of idle heterogeneous machines within a network. It is an efficient and reliable high-throughput computer system that manages the dynamic and heterogeneous resources in a computer network environment. CONDOR uses a ClassAd implementation that works by matching a resource request from a client (which defines the resource requirements needed to run a job) to a resource offer from a machine in the network (which advertises its resources such as an available memory, computer type, etc.) See figure 15 for a graphical presentation of ClassAd. To preserve the computational work that has been performed on each machine, CONDOR offers a checkpointing and migration feature [52]. CONDOR periodically checkpoints a job during the execution of the program to protect the computational results in the event of an owner of the machine returning or a system failure such as a machine crashing or a power outrage.

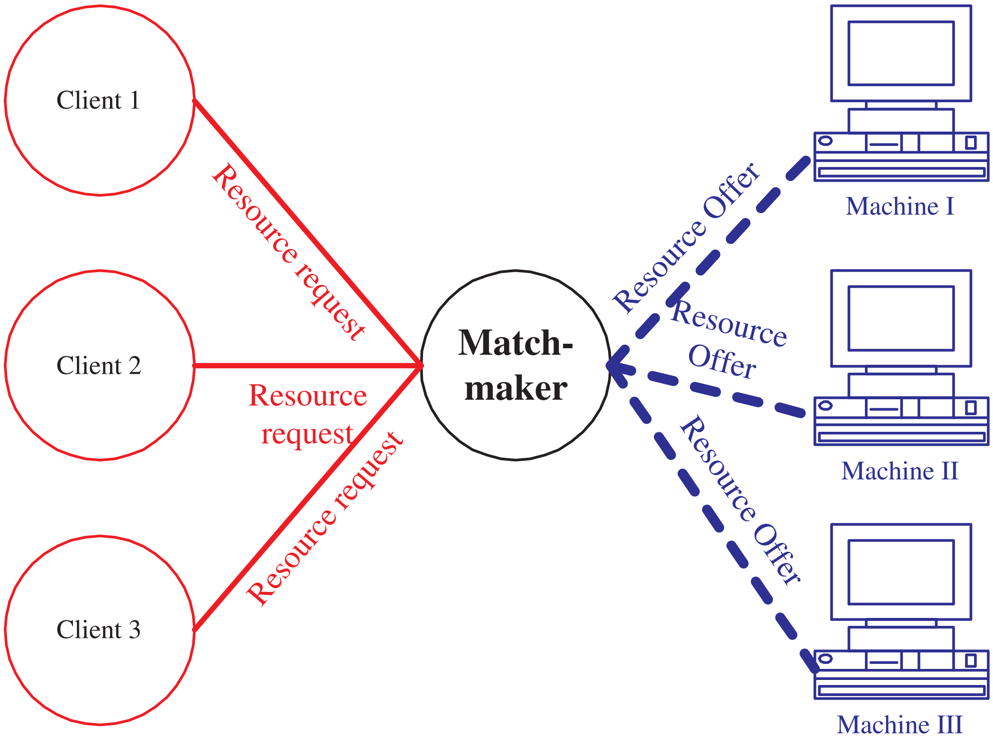

Figure 15: ClassAd diagram that represents the CONDOR resource management.

In section 2.3, we explained how our gradient approximation based on a quadratic model fits in this framework. This quadratic model requires at least $\frac{n(n+1)}{2}+n+1$ sampling simulation runs. These work loads can take an enormously long time to compute serially. Because these points are randomly and independently generated, it is a perfect fit for the master-worker paradigm of parallel computation. In this paradigm, the master generates the work for each worker (in this case simulation runs at different point) and waits for the results from each worker. The master will wait for a preset amount of time. If some workers do not finish the jobs within this time period, then the master will cancel the current running jobs and resubmit the incomplete job to the workers again. After all workers have completed their tasks, the master then reports the simulation results and stops.

The master-worker implementation [34] as shown in figure 16 can be easily adapted

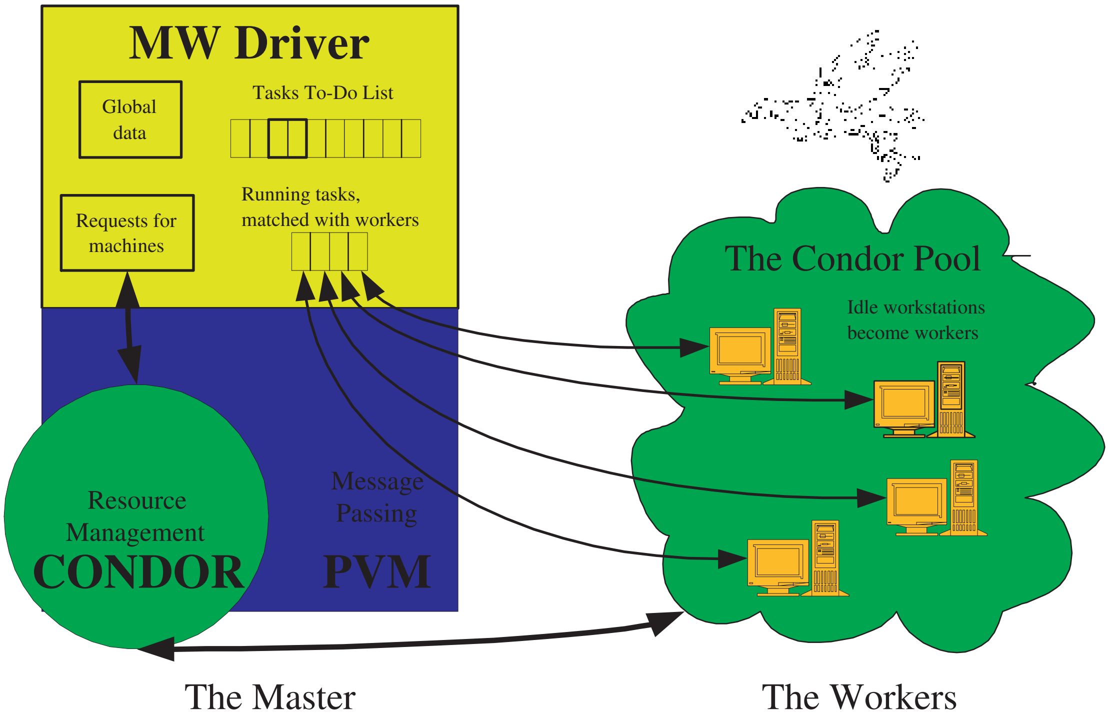

Figure 16: The Relationship between the MWDriver, Condor, and PVM , excerpt from [34]

to our situation. We have implemented a Master-Worker Condor-PVM server based on the available MW libraries written at University of Wisconsin-Madison that will be described in section 3.4. However, we first detail a more straightforward approach to utilize a collection of workstations in parallel.

3.2 Job parallel simulation optimization

Our initial master-worker implementation makes use of CONDOR directly to distribute the computation throughout the heterogeneous of network machines. For computing one derivative, simulation runs are carried out in parallel using a single CONDOR submit file. The master implementation creates input files which contain the simulation points and submits them as separate jobs. The master then checks the completion of all simulation runs and collects them via output files. In order to take advantage of executing the simulations on different types of machines based on their availabilities within the CONDOR environment, we have to supply a different binary executable simulation program based on each architecture and operating system type as a batch-ready program that accepts an input file and generates an output file after it is complete. This process does not require any simulation source code and it can be compiled using an appropriate compiler on a specific architecture and operating system. However, during the linking process, we need to re-link the binary executable program using the 'condor_compile' statement.

For example, the simulation object file, 'simulation.o', can be compiled from Fortran 77, but it must be re-linked as

```
condor_compile f77 -o qso_INTEL_LINUX.exe simulation.o
```

Our standard CONDOR submit file will request this program to be executed as

```
qso_INTEL_LINUX.exe conqso_0.ifn conqso_0.ofn
```

where 'conqso_0.ifn' is the simulation input file and 'conqso_0.ofn' is the simulation output file.

```
NUMRUNS = 7
universe = standard
requirements = ((ARCH == "SUN4u" && OPSYS == "SOLARIS26") \
             || (ARCH == "INTEL" && OPSYS == "SOLARIS26") \
             || (ARCH == "INTEL" && OPSYS == "LINUX"))
executable = qso_$$(ARCH)_$$(OPSYS).exe
transfer_input_files = qso.cfn, conqso_$(Process).ifn
transfer_output_files = conqso_$(Process).ofn
arguments = conqso_$(Process).ifn conqso_$(Process).ofn
log = condor_qso.log
notification = Error
queue $(NUMRUNS)
```

An example of a CONDOR submit file containing 7 simulation runs is shown above. Each simulation run must put the parameters and decision variables in the input files of the

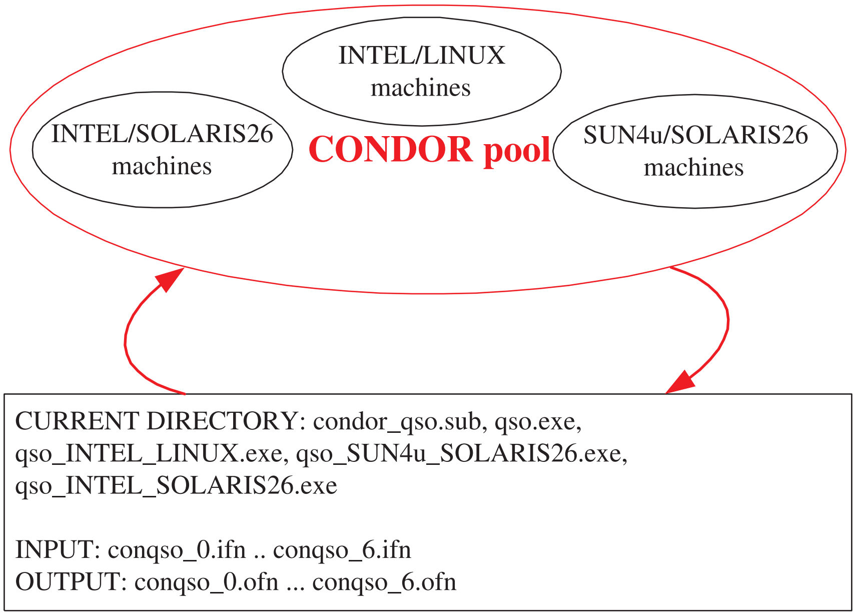

Figure 17: CONDOR multiple architectures diagram.

form `conqso_$(Process).ifn` where `$(Process)` represents a number from 0 to 6. After all jobs are complete, the output files of the form `conqso_$(Process).ofn` will contain the output of each simulation run from 0 to 6. We submit all the jobs in one cluster under the 'standard' universe to take advantage of the checkpointing and migration features. Note that if the job is interrupted during its execution, then it will continue executing from the last checkpoint on the next machine having the same architecture. Furthermore, once a job is started on one machine type, the checkpointing and migration must maintain the computer architecture type.

For this submit file to be executed properly, the current directory must contain three binary executable files; qso_SUN4u_SOLARIS26.exe, qso_INTEL_SOLARIS26.exe and qso_INTEL_LINUX.exe, see figure 17.

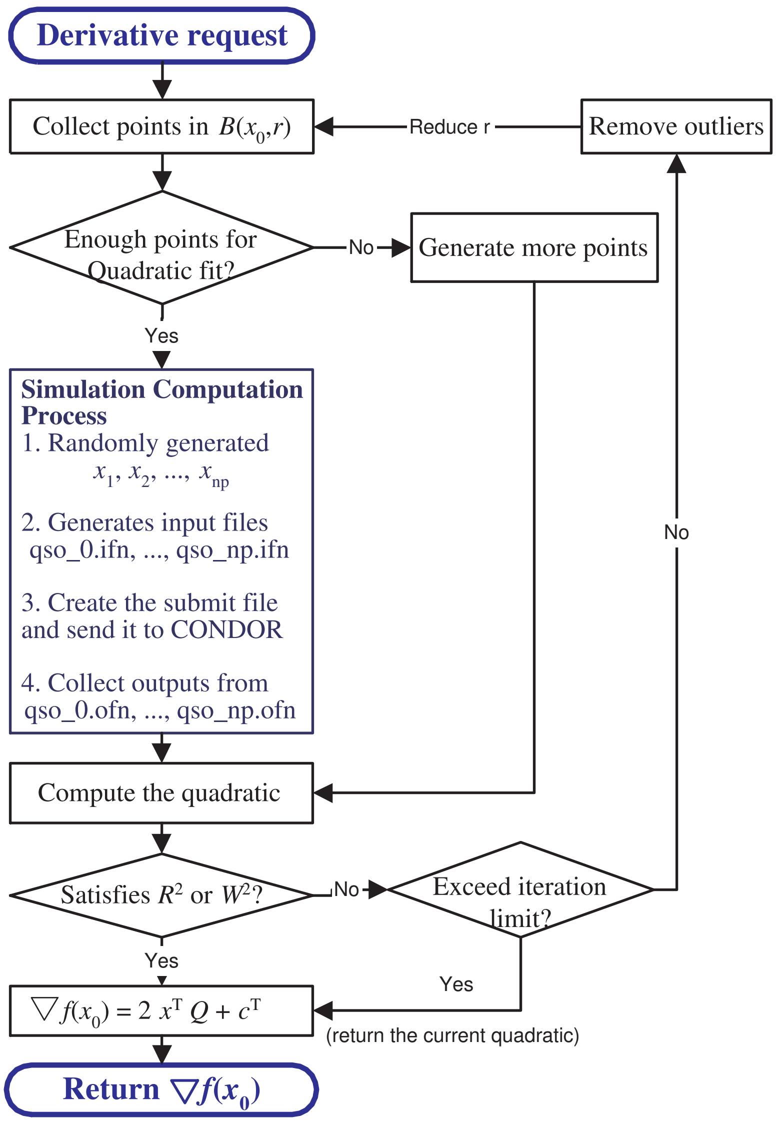

Figure 18: Derivative computation flow chart.

The program computes simulation values as follows:

- It creates all input files. Each file contains fixed parameters and values of decision variables for a single simulation point.
- It generates a submit file which contains the number of simulation computation requests for the CONDOR pool as described above.

- It makes a system call to 'condor_submit' to submit to the CONDOR pool.
- It waits until jobs are completed or jobs are removed from the CONDOR pool.
- It continues the gradient approximation using available computed simulation points.

After all simulation results are returned from CONDOR, our program applies the least squares approximation to construct the quadratic model. The rest of the process is similar to the serial version in section 2.4.2. The graphical implementation is given in Figure 18.

3.3 Parallel computational results from CONDOR

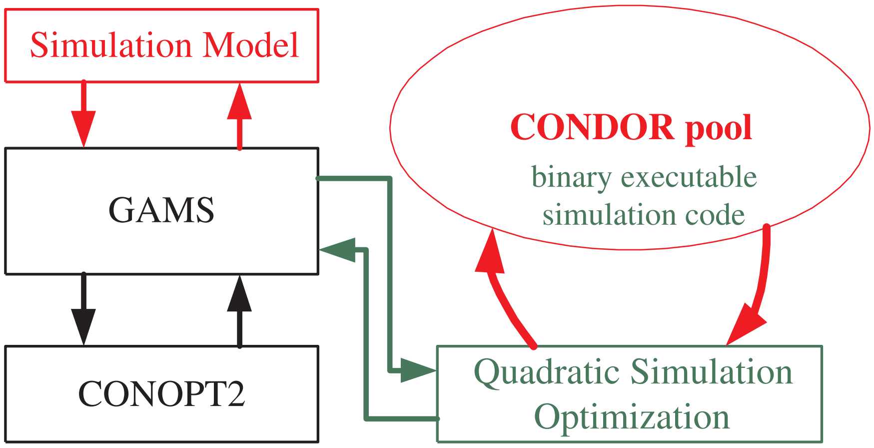

Figure 19: The CONDOR simulation optimization overview

Figure 19 shows our schema of the QSO program which takes advantage of parallelism using CONDOR. in this diagram, GAMS is a mediator that communicates between a practitioner model, a nonlinear programming solver, and our parallel QSO module. After a pracitioner submits the model to GAMS, GAMS interactively calls the nonlinear programming solver and supplies function and derivative computations. GAMS will interact with the QSO module only when the function or derivative request of an external equation is called from the nonlinear programming solver.

When the nonlinear programming solver requests a value of the function written using GAMS syntax, GAMS internally evaluates the function and passes it to the nonlinear programming solver. If the request is for the external function, then GAMS calls the QSO module via a dynamic link library or a shared library. The QSO module checks if the function has been called before and makes a system call to a simulation module as necessary. The result is then returned to the QSO module and reports back to GAMS. The computation of the derivative is different from the function computation. For the derivative evaluation of an equation written in GAMS, automatic differentiation is carried out. This gives a reliable derivative and it is applicable because the equation source code is available. However, if the derivative evaluation is called for an external equation, GAMS calls the QSO module and passes variable values using the mapping that is described in section 2.2 via the dynamic link library or the shared library. The QSO module generates random sampling points around the requested point using the current radius. It generates a simulation option file, a simulation configuration file and the simulation input files. Then it submits a CONDOR file to the CONDOR scheduler for the parallel computation using heterogeneous machines. After all computations are completed, it applies the least squares approximation to fit the quadratic model. If the quadratic model shows an acceptable fit for these simulation points, then it returns the derivative of the quadratic model. Otherwise, it reduces the current radius around that point, resampling additional points if necessary and fits the quadratic model again.

We solve the same tandem production line problems appeared in section 2.5.3 based on longer simulation length. Each simulation collects data from a 100,000 products with initial warm-up of 1,000 products. Based on the same random seed, we rerun the solution

Table 9: Objective comparisons between the SRO, BSO, serial QSO, and CONDOR QSO

Case	*SRO*	*BSO*	*QSO*	
			serial	*CONDOR*
1a	7.7132	7.7128	7.7132	7.7132
1b	7.7232	7.7231	7.7231	7.7231
2a	0.9729	0.9729	0.9729	0.9729
2b	0.9983	0.9983	0.9983	0.9983
3a	0.7318	0.7319	0.7318	0.7318
3b	0.7336	0.7383	0.7335	0.7335
4a	0.3956	0.3956	0.3956	0.3956
4b	0.3962	0.3962	0.3962	0.3963
5a	0.3496	0.3497	0.3487	0.3487
5b	0.3475	0.3474	0.3457	0.3457

results from Plambeck paper [60] both SRO and BSO. Table 9 shows the comparable results from QSO and CONDOR QSO with the SRO and BSO method.

Table 10: Timing comparison of the serial QSO and CONDOR QSO method

Case	*Serial QSO*		*CONDOR QSO*		
	Total runs (sim.)	*Usage time (sec.)*	*Total runs (sim.)*	*Master time (sec.)*	*CONDOR time (sec.)*
1a	239	515	239	118	2527
1b	256	581	256	106	2315
2a	176	51	176	13	2742
2b	149	43	281	30	3910
3a	801	4251	749	1552	2618
3b	955	5168	910	1009	13061
4a	3648	46824	6528	28555	38612
4b	10988	154400	1641	1126	22904
5a	4355	4312	4092	1137	29150
5b	11408	15514	6397	3496	32943

Table 10 shows comparisons between the serial program and the parallel program using CONDOR. Unfortunately, these computational results are disappointing since each time a derivative evaluation is requested, the matching algorithm requests new machines.

These requests can take a long time to be satisfied resulting in the poor results show in table 10. To alleviate this problem we have developed a Master-Worker server that we describe in 3.4.

3.4 Master-Worker condor-PVM server

Due to the additional overhead of the CONDOR scheduling server, the wall clock time for solving a simulation optimization is dominated by the total waiting time for resource provision from CONDOR. To remedy this situation, we use a Master-Worker server paradigm which is adopted from [34]. The key idea in this approach is to maintain hold over a cluster of machines throughout the complete optimization process, rather than requesting new computational resources 'on-the-fly'.

Typical use of the Master-Worker framework generates either an initial list of tasks to perform, or generates new tasks based on the solution of previous tasks (eg. branch and bound procedures). Once the task list is empty, the program terminates. In contrast, our server is assumed to be persistent, only terminating when a particular flag is set in the file system. In order to effect this, we have implemented a "idlewait" task that just spins for a certain amount of time. Thus whenever the server has no work (simulations) to perform, it just waits idling. Whenever new work is generated, the server already has enough computational resources to perform many tasks in parallel.

Figure 20 shows how the Master-Worker Condor-PVM server communicates with the GAMS system. The Master-Worker server runs as a separate program that monitors a request of computations from the file system. A practitioner executes a model in GAMS which passes the control to the nonlinear solver. If the nonlinear solver requests a function or derivative evaluation of the external equation, then it passes a request to QSO module. The QSO module writes an input file that contains all simulation points, then

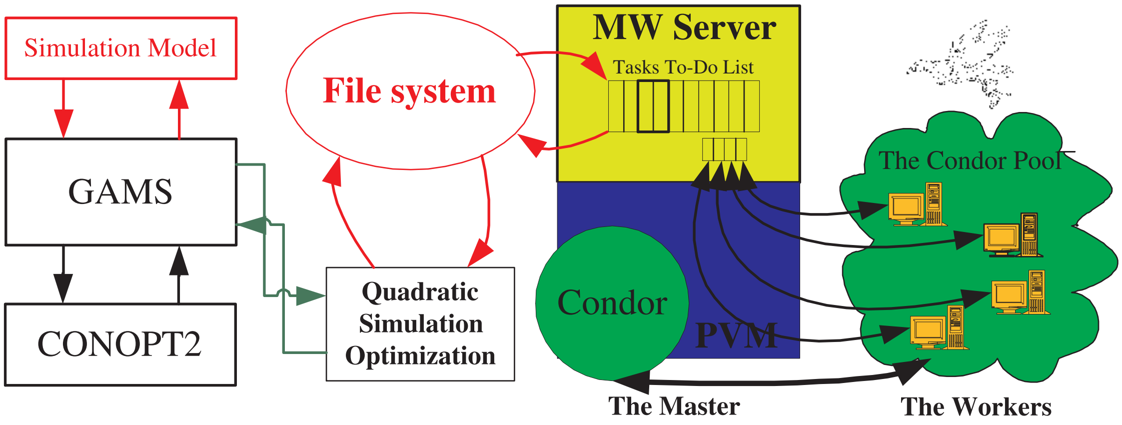

Figure 20: Master-Worker server and GAMS.

updates the status file to trigger the Master-Worker server. After the Master-Worker server receives a request, it reads the input file and assigns all points to the Tasks To-Do List. It takes care of requesting workers from the CONDOR pool, assigning tasks to workers, and reassigning tasks if workers fail. Upon completing the last task in the Tasks To-Do List, the Master-Worker server writes the output to a file and then updates the status file to inform the QSO module, which has been waiting for the output simulations. The QSO module passes the result back to the nonlinear solver.

Table 11 shows a comparison of the total running time between the serial version of QSO and the MW-server version of QSO. The *Excess* column shows the number of simulation calls for the gradient computation that can be saved if the parallel computations are used. The *Parallel saving* column is computed from the average saving time of the *Excess* simulation by multiplying the *Excess* column with the *Runs* column and dividing by the *Time* column. Note that we gain larger saving using the MW-server rather than regular parallel computations because we allow the more powerful computational machines to execute the simulation runs.

Figure 21 shows that we gain more than 100% saving on problems 1a, 1b, 2a, and 2b which are distributed to the faster simulation runs on the more powerful machines. For

Table 11: Timing comparison of the serial QSO and QSO with MW server.

Case	*Serial QSO*					*MW QSO*			
	Runs (sim.)	*Grad.*	*Time (sec.)*	*Excess (sim.)*	*Parallel saving*	*Runs (sim.)*	*Grad.*	*Time (sec.)*	*Saving (sec.)*
1a	432	20	589	138	188.15	432	20	314	275
1b	424	20	584	165	227.26	424	20	275	309
2a	248	16	36	77	11.18	248	16	20	16
2b	316	23	48	101	15.34	316	23	24	24
3a	796	45	2983	280	1049.30	862	55	2626	357
3b	1490	238	4963	497	1655.44	1504	45	3583	1380
4a	4016	204	32075	1718	13721.33	5772	402	19885	12190
4b	4840	214	40641	1942	16306.78	6650	309	17856	22785
5a	10712	435	5630	4736	2489.14	12904	209	3888	1742
5b	8170	429	4500	3306	1820.93	7824	188	2875	1625

problem 3a, 3b, 4a, 4b, 5a and 5b, the CONOPT computation show different path to the optimal solution because of the heterogeneous computation of different machine architectures. Nevertheless, we still gain the total running time for all these cases. Problem 3a, 4a and 4b, the MW-server QSO uses more derivative computations than the serial QSO. However, it achieves the better running time due to the fact that more powerful machines are used for these runs which are requested by the MW-server routine. Problem 3b, 5a and 5b, the MW-server QSO uses less derivative computations so that we would expect the faster running time.

As table 11 shows, we gain a maximum of 50% total running time because the simulation runs that can be saved are about 50% of the total simulation runs. In order for our MW-server algorithm to be more effective, the nonlinear programming solver that it uses should request more gradient evaluations during the solving process. Even for a nonlinear programming solver that does not require derivative computation, we can gain the faster running time due to the availability of faster machines in the CONDOR pool.

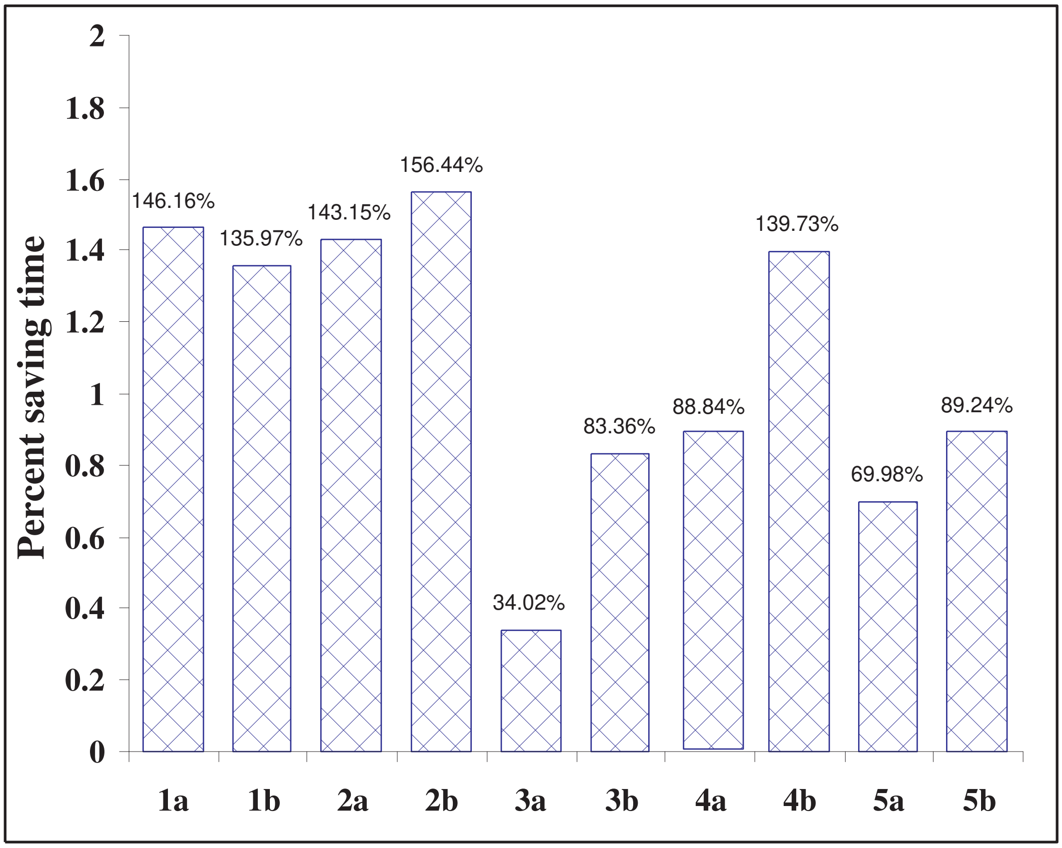

Figure 21: Percentage of timing saving of the MW-server QSO comparing to the maximum saving time for parallel computations of the same machine.

In all these cases, we would not expect the running time of MW-server QSO to be shorter than the running time of the serial QSO because one machine is always available at any time.

Chapter 4

Nonlinear programming via MCP

4.1 Mathematical formulation

In this chapter, we solve the following constrained nonlinear program (NLP)

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g(x) \le 0,\ h(x) = 0,\ x \in B, \end{array} \tag{9}$$

where $f : I\!R^n \mapsto I\!R$, $g : I\!R^n \mapsto I\!R^m$ and $h : I\!R^n \mapsto I\!R^p$ are twice continuously differentiable, and $B := \{x \in I\!R^n | r \le x \le s\}$ with $r_i \in [-\infty, \infty]$ and $s_i \in [r_i, \infty]$. Let $S := \{x \in B | g(x) \le 0, h(x) = 0\}$ denote the feasible region. We will focus on finding a point that satisfies the first-order conditions of (9).

4.1.1 The first-order conditions of NLP

The concept of the Lagrangian function and the Lagrange multipliers play a crucial role in defining a first-order point of NLP. The Lagrangian function is a weighted summation of the objective function and the constraint functions, defined as follows

$$L(x, \lambda, \nu) := f(x) - \lambda^T g(x) - \nu^T h(x), \tag{10}$$

where λ and ν denote the Lagrange multipliers (dual variables) corresponding to the inequality and equality constraints, respectively.

The first-order necessary conditions for NLP are

$$\begin{aligned} 0 &\in \nabla_x L(x, \lambda, \nu) + N_B(x) \\ 0 \geq \lambda &\perp g(x) \leq 0 \\ h(x) &= 0, \end{aligned} \tag{11}$$

where $N_B(x) = \{z \in I\!R^n | (y-x)^T z \leq 0, \forall y \in B\}$ is the normal cone [67] to B at x.

In the case that r_i or s_i is finite, the definition of the normal cone allows the first equation of (11) to be rewritten in the following manner. If $x_i = r_i$, then

$$(\nabla_x L(x, \lambda, \nu))_i \geq 0,$$

while if $x_i = s_i$, then

$$(\nabla_x L(x, \lambda, \nu))_i \leq 0$$

and for any values of r_i and s_i, if $r_i < x_i < s_i$, then

$$(\nabla_x L(x, \lambda, \nu))_i = 0.$$

These conditions coupled with a regularity condition on the point x establish the necessary conditions for NLP which are normally called the Karush-Kuhn-Tucker (KKT) conditions [45, 49]. Whenever the Hessian matrix of the Lagrangian function is positive definite at (x^*, λ^*, ν^*), the first-order conditions are also sufficient for x^* to be a strict local minimizer of NLP.

4.1.2 Primal-dual formulation of NLP

The standard Mixed Complementarity Problem (MCP) is defined as the problem of finding a point $z \in I\!R^n$ inside the box $B = \{z | -\infty \leq l < z < u \leq \infty\}$ that is complementary to a nonlinear function $F : I\!R^n \to I\!R^n$. We assume without loss of generality that $l_i < u_i$ for all $i = 1, 2, \ldots, n$.

The point z is complementarity to $F(z)$ when

$$\begin{array}{llll} \text{either} & z_i = l_i & \text{and} & F_i(z) \geq 0 \\ \text{or} & z_i = u_i & \text{and} & F_i(z) \leq 0 \quad \text{for } i = 1, \ldots, n \\ \text{or} & l_i < z_i < u_i & \text{and} & F_i(z) = 0. \end{array}$$

If $l \equiv -\infty$ and $u \equiv +\infty$, MCP becomes the problem of finding a zero of a system of nonlinear equations, that is to find $z \in I\!R^n$ such that $F(z) = 0$, while if $l = 0$ and $u \equiv +\infty$, the problem is the Nonlinear Complementarity Problem (NCP) of finding $z \in I\!R^n$ such that $z_i \geq 0, F_i(z) \geq 0$ and $z_i F_i(z) = 0$, for all $i = 1, \ldots, n$. The latter property $z_i F_i(z) = 0$ is often called complementarity between z_i and $F_i(z)$.

Let z be composed of the primal variable x and the dual variables λ and ν of NLP. The nonlinear MCP function can be written as a vector function of the first-order derivative evaluation of the Lagrangian function with respect to the corresponding primal and dual variables, that is

$$F(z) := \begin{bmatrix} \nabla_x L(z) \\ -\nabla_\lambda L(z) \\ -\nabla_\nu L(z) \end{bmatrix}.$$

The nonlinear MCP model is to find $z = (x, \lambda, \nu) \in I\!R^q$ where $q = n + m + p$ that is complementary to the nonlinear vector function F from $I\!R^q \mapsto I\!R^q$ given above along with lower bounds l and upper bounds u

$$F(z) = \begin{bmatrix} \nabla_x L(z) \\ g(x) \\ h(x) \end{bmatrix}, l := \begin{bmatrix} r \\ -\infty \\ -\infty \end{bmatrix}, u := \begin{bmatrix} s \\ 0 \\ +\infty \end{bmatrix}.$$

Here $\nabla_x L(z) = \nabla_x f(x) - \lambda^T \nabla_x g(x) - \nu^T \nabla_x h(x)$

$$= \nabla_x f(x) - \textstyle\sum_{i=1}^m \lambda_i \nabla_x g_i(x) - \sum_{j=1}^p \nu_j \nabla_x h_j(x).$$

By comparing MCP to KKT, we can see that this formulation is equivalent to the first-order conditions of our original NLP program. This allows us to solve the NLP problem using an MCP solver, which is the subject of section 4.3.

4.2 The PATH solver and the merit function

The PATH solver [18] is a nonsmooth Newton type algorithm [65] which finds a zero of the normal map [64]

$$F_{+}(x) := F(\pi(x)) + x - \pi(x),$$

where $\pi(x)$ is the closest point in B to the variable x in the Euclidean norm. It is well known [64] that finding a zero of this normal map is equivalent to solving MCP. In particular if x is a zero of the normal map, then $\pi(x)$ solves MCP, while if z solves MCP then $z - F(z)$ is a zero of the normal map.

4.2.1 Overview of the PATHNLP algorithm

The essential idea of the code is to linearize the normal map $F_{+}(x)$ about the current iterate to obtain a piecewise linear map whose zero is sought using a homotopy approach [19]. To monitor progress in the nonlinear model, a nonmonotone path-search is used [62]. Recent extensions [23] have introduced a new merit function Ψ to be used in conjunction with the code.

The following pseudo code shows the main algorithm steps of the PATH solver to find a KKT point:

Loop until $\Psi(x)$ is less than convergence tolerance {

Solve the linearization of the MCP problem to obtain the Newton point;

Search the path between the current point and the Newton point.

If the new point gives rise to a better merit function value then accept this new point.

Otherwise use the merit function to find a descent direction and search along this direction for a new point.

}

Details on the solution of linearization and the path-search mechanism can be found in [25, 18]. In this chapter, we just indicate the changes specific to solving NLP's. The Newton-type PATH solver uses the Jacobian matrix of the MCP function to find its path-searching direction. In the above context, the Jacobian matrix is computed by finding the derivative of the MCP function. It uses the first and second order derivatives of the original NLP objective function and constraints as

$$\nabla_z F(z) := \begin{bmatrix} \nabla^2_{xx} L(x, \lambda, \nu) & -\nabla^T_x g(x) & -\nabla^T_x h(x) \\ \nabla_x g(x) & 0 & 0 \\ \nabla_x h(x) & 0 & 0 \end{bmatrix}, \tag{12}$$

where $\nabla^2_{xx}L(x,\lambda,\nu) = \nabla^2_{xx}f(x) - \sum_{i=1}^{m}\lambda_i\nabla^2_{xx}g_i(x) - \sum_{j=1}^{p}\nu_j\nabla^2_{xx}h_j(x)$.

4.2.2 The merit function for the PATH solver

The most recent version of the PATH solver [23] does not use the residual of the normal map for a merit function. Instead, it utilizes the Fischer-Burmeister function [27] defined as the mapping $\phi : I\!R^2 \to I\!R$,

$$\phi(p,q) := \sqrt{p^2+q^2} - p - q,$$

where p and q are scalar variables. This function exhibits the complementarity property when the function value is zero, that is

$$\phi(p,q) = 0 \text{ if and only if } p \geq 0, q \geq 0 \text{ and } pq = 0.$$

For the MCP problem, the merit function used is $\Psi : I\!R^n \to I\!R$,

$$\Psi(x) := \frac{1}{2}\psi(x)^T\psi(x),$$

where $\psi(x)$ is the Fischer operator defined in [10] from $I\!R^n$ to $I\!R^n$ that maps x_i and $F_i(x)$ as parameters to the Fischer-Burmeister function component-wise as follows:

$$\psi_i(x) := \begin{cases} \phi(x_i - l_i, F_i(x)) & \text{if } -\infty < l_i \leq x_i < +\infty, \\ -\phi(u_i - x_i, -F_i(x)) & \text{if } -\infty < x_i \leq u_i < +\infty, \\ \phi(x_i - l_i, \phi(u_i - x_i, -F_i(x))) & \text{if } -\infty < l_i \leq x_i \leq u_i < +\infty, \\ -F_i(x) & \text{if } -\infty < x_i < +\infty. \end{cases} \tag{13}$$

This merit function is nonnegative and is zero at the solution point. A key feature of this merit function is its continuously differentiability. It allows gradient steps to be used when the path-searching direction does not lead to a descent direction.

The nonlinear MCP function contains only the first-order derivatives of the objective function and constraints as in section 4.1.2. The formulation exhibits the deficiency of finding KKT points for NLP. To try to avoid this deficiency, we introduce a new merit function for the PATH solver that explicitly incorporates the objective function. We now describe the implementation of the new merit function and give some computational results in section 4.4.

The PATH solver uses a merit function to find a gradient descent direction when its Newton direction fails to find a descent direction. It uses the residual function $\Psi(x)$ to identify the stopping criteria. We define a new merit function for the PATH solver applied to NLP's which is a weighted average of the residual function Ψ and the objective function f as

$$\varphi(x) = (1-\gamma)\Psi(x) + \gamma f(x), \tag{14}$$

where $\gamma \in [0, 1]$.

When γ is equal to zero, $\varphi(x) = \Psi(x)$ which is the original PATH solver that finds the first-order conditions of the NLP problem. For $\gamma > 0$, the objective function affects the search direction. However, if the weight on the objective function is 1, then the merit function is simply the objective function so a solution is not guaranteed to satisfy the first-order conditions. With appropriate choice of γ, our new merit function guides the path-searching algorithm to escape KKT points that are not local minimizers of the original NLP. After our experimentation with the value of γ, we decided to take a fixed value of $\gamma = 0.3$ for the purposes of the results given in section 4.4.

In the next section, we show how the NLP model in AMPL is automatically modified and transformed into the MCP formulation. The MCP function evaluation and its Jacobian evaluation are specified in more detail.

4.3 NLP solver via AMPL

To solve the NLP problem in AMPL, a user can specify the complementarity formulations directly using the AMPL language [22]. This requires a modeler to write down explicitly the first-order conditions as detailed in section 4.1.2. This is very cumbersome and prone to error. In this chapter, we propose to use the AMPL solver library to take an NLP specified directly in AMPL and form the required F and its Jacobian matrix for the PATH solver within the solver link. This means that a modeler simply has to change the solver name in order to use the approach outlined in this chapter.

4.3.1 MCP formulation from AMPL

The NLP problem passed to a solver from the AMPL environment is defined as

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & a \leq c(x) \leq b \\ & r \leq x \leq s, \end{array} \tag{15}$$

where $f : I\!R^n \mapsto I\!R, c : I\!R^n \mapsto I\!R^m$ with $a, b \in I\!R^m$ and $x, r, s \in I\!R^n$.

We now show how to recover the NLP format as described in section 4.1 from the data given above. We define five mutually exclusive index subsets of an index set $I = \{1, 2, \ldots, m\}$ of the constraint function c as

$$\begin{aligned} \mathcal{L} &:= \{i \in I | -\infty < a_i \text{ and } b_i \equiv +\infty\} \\ \mathcal{U} &:= \{i \in I | a_i \equiv -\infty \text{ and } b_i < +\infty\} \\ \mathcal{E} &:= \{i \in I | -\infty < a_i = b_i < +\infty\} \\ \mathcal{R} &:= \{i \in I | -\infty < a_i < b_i < +\infty\} \\ \mathcal{F} &:= \{i \in I | a_i \equiv -\infty \text{ and } b_i \equiv +\infty\}, \end{aligned} \tag{16}$$

where $\mathcal{L}$ is the index set of lower bound constraints, $\mathcal{U}$ is the index set of upper bound constraints, $\mathcal{E}$ is the index set of equality constraints, $\mathcal{R}$ is the index set of range constraints, and $\mathcal{F}$ is the index set of free constraints.

The NLP model from AMPL is therefore rewritten as

$$
\begin{array}{llcll}
\text{minimize} & & f(x) & & \\
\text{subject to} & a_i \leq & c_i(x) & & i \in \mathcal{L} \\
& & c_i(x) & \leq b_i & i \in \mathcal{U} \\
& & c_i(x) & = a_i & i \in \mathcal{E} \\
& a_i \leq & c_i(x) & \leq b_i & i \in \mathcal{R} \\
& & c_i(x) & \text{is free} & i \in \mathcal{F} \\
& r \leq & x & \leq s. &
\end{array}
\tag{17}
$$

Define $y \in I\!R^{|\mathcal{R}|}$ as artificial variables for each range constraint, where $|\mathcal{R}|$ is the number of range constraints. Then by dropping the free constraints, the model is equivalent to

$$
\begin{array}{llll}
\text{minimize} & f(x) & & \\
\text{subject to} & a_i - c_i(x) & \leq 0 & i \in \mathcal{L} \\
& c_i(x) - b_i & \leq 0 & i \in \mathcal{U} \\
& c_i(x) - a_i & = 0 & i \in \mathcal{E} \\
& c_i(x) - y_{j_i} & = 0 & i \in \mathcal{R} \\
& a_i \leq y_{j_i} \leq b_i & & i \in \mathcal{R} \\
& r \leq x \leq s, & &
\end{array}
\tag{18}
$$

where j_i is the index from 1 to $|\mathcal{R}|$, corresponding to the order of index $i \in \mathcal{R}$.

We write the constraint function g and h of NLP as

$$
g(x) = \begin{cases} a_i - c_i(x) & \text{if } i \in \mathcal{L} \\ c_i(x) - b_i & \text{if } i \in \mathcal{U} \end{cases}
$$

and

$$h(x) = \begin{cases} c_i(x) - a_i & \text{if } i \in \mathcal{E} \\ c_i(x) - y_{j_i} & \text{if } i \in \mathcal{R}. \end{cases}$$

The new Lagrangian function for this model is

$$\begin{aligned} L(x, \lambda, \nu, y) = \; & f(x) - \lambda_{\mathcal{L}}^T(a_{\mathcal{L}} - c_{\mathcal{L}}(x)) - \lambda_{\mathcal{U}}^T(c_{\mathcal{U}}(x) - b_{\mathcal{U}}) - \\ & \nu_{\mathcal{E}}^T(c_{\mathcal{E}}(x) - a_{\mathcal{E}}) - \nu_{\mathcal{R}}^T(c_{\mathcal{R}}(x) - y). \end{aligned}$$

Defining $\lambda = (\lambda_{\mathcal{L}}, \lambda_{\mathcal{U}})$ and $\nu = (\nu_{\mathcal{E}}, \nu_{\mathcal{R}})$, the corresponding MCP model is to find $z = (x, \lambda, \nu, y) \in I\!R^q$ (where $q = n + m + |\mathcal{R}|$) that is complementary to a nonlinear vector function F from $I\!R^q \rightarrow I\!R^q$ defined as

$$F(z) := \begin{bmatrix} \nabla_x L(z) \\ a_{\mathcal{L}} - c_{\mathcal{L}}(x) \\ c_{\mathcal{U}}(x) - b_{\mathcal{U}} \\ c_{\mathcal{E}}(x) - a_{\mathcal{E}} \\ c_{\mathcal{R}}(x) - y \\ \nu_{\mathcal{R}} \end{bmatrix},$$

where $\nabla_x L(z) = \nabla_x f(x) - \lambda^T \nabla_x g(x) - \nu^T \nabla_x h(x)$, and

$$\begin{bmatrix} r \\ -\infty \\ -\infty \\ a_{\mathcal{R}} \end{bmatrix} \leq z = \begin{bmatrix} x \\ \lambda \\ \nu \\ y \end{bmatrix} \leq \begin{bmatrix} s \\ 0 \\ +\infty \\ b_{\mathcal{R}} \end{bmatrix}.$$

4.3.2 Solver links in AMPL

AMPL executes the NLP solver as a separate program and communicates with it using the file system. Files with extension `.nl` contain a description of the model whereas files with extension `.sol` contain a termination message and the final solution written by the solver. The AMPL system uses information from these files to allocate space, set its ASL structure and set global variable values. These values are used to identify the problem dimension, the value of objective function at the current point, the gradient evaluation, the constraint evaluation and its Jacobian and Hessian sparse matrices.

Useful global variables are

`n_var` as the total number of variables,

`n_obj` as the total number of objective functions,

`n_con` as the total number of constraints,

`nzc` as the number of nonzeros in the Jacobian matrix and

`nzo` as the number of nonzeros of the objective gradient.

The ASL structure is made up of two main components, `Edagpars` and `Edaginfo`. The `Edagpars` contains information to evaluate the objective function, constraint functions and their first and second order derivatives. The `Edaginfo` contains the upper and lower bounds, the initial point, the compressed column structure of the Jacobian matrix of the constraint functions, the pointer structure of the first order derivatives of the objective function and constraints, and information about the NLP problem. For a complete listing of all global variables and the ASL structure, the reader should consult the AMPL manual [28].

A detailed description of our implementation, called `pathnlp`, now follows. After the `solve` command is invoked in AMPL, the AMPL system generates associated NLP problem files and communicates to the `pathnlp` solver. This solver, written in the C language, automatically constructs the primal-dual formulation of the original NLP problem. It calls the PATH solver with additional options if necessary. The PATH solver runs and returns the status of the solution point via the `Path_Solved` variable and the final solution z using the `Path_FinalZ(p)` routine. The link returns these results to the AMPL system by calling `write_sol`. AMPL reports the solution back to the user who further analyzes and manipulates the model.

We now give details of how F and $\nabla_z F$ are evaluated in the link.

- Our program allocates the ASL structure by calling `ASL_alloc` with parameter `ASL_read_pfgh` which requests the AMPL to generate all first-order and second order derivatives of the objective function and constraints. In addition, the flag, `want_xpi0 = 1` is set to 1 to request the initial point. The flag, `want_deriv = 1` is set to 1 to request Jacobian evaluations and Hessian evaluations.

- Our program initializes all NLP variables by calling `getstub`. It calls `jacdim` to obtain information about the Jacobian and Hessian of the objective function and constraints.

- Our program defines the MCP variable z as (x, λ, ν, y) and sets up the lower bound as $(r, -\infty, -\infty, a_{\mathcal{R}})$ and the upper bound as $(s, 0, +\infty, b_{\mathcal{R}})$.

- The function evaluation of the MCP model is defined as

$$F(z) := \begin{bmatrix} \nabla_x L(z) \\ a_{\mathcal{L}} - c_{\mathcal{L}}(x) \\ c_{\mathcal{U}}(x) - b_{\mathcal{U}} \\ c_{\mathcal{E}}(x) - a_{\mathcal{E}} \\ c_{\mathcal{R}}(x) - y \\ \nu_{\mathcal{R}} \end{bmatrix}. \tag{19}$$

  The value of this function at the current point is kept in the vector F. To compute $\nabla_x L(z) = \nabla_x f(x) - \lambda^T \nabla_x g(x) - \nu^T \nabla_x h(x)$, the program first evaluates $\nabla_x f(x)$ at the current point by calling `objgrd`. It retrieves the sparse Jacobian matrix of c by calling `jacval` and uses `Cgrad` as the sparse matrix structures. This produces values of $c(x)$. Then it multiplies the sparse Jacobian matrix with the corresponding Lagrange multipliers and subtracts these to $\nabla_x f(x)$. The rest of the vector is computed by calling `conval` and using the appropriate multipliers of 1, -1 or 0 to generate the vector F. Then it copies the values of $\nu_{\mathcal{R}}$ for the last $|\mathcal{R}|$ elements.

- The Jacobian evaluation of this MCP is given as

$$\begin{bmatrix} \nabla^2_{xx} L(z) & +\nabla_x c_{\mathcal{L}}(x) & -\nabla_x c_{\mathcal{U}}(x) & -\nabla_x c_{\mathcal{E}}(x) & -\nabla_x c_{\mathcal{R}}(x) & 0 \\ -\nabla_x c_{\mathcal{L}}(x) & 0 & 0 & 0 & 0 & 0 \\ +\nabla_x c_{\mathcal{U}}(x) & 0 & 0 & 0 & 0 & 0 \\ +\nabla_x c_{\mathcal{E}}(x) & 0 & 0 & 0 & 0 & 0 \\ +\nabla_x c_{\mathcal{R}}(x) & 0 & 0 & 0 & 0 & -I \\ 0 & 0 & 0 & I & 0 & 0 \end{bmatrix}. \tag{20}$$

This computation uses the Hessian of the Lagrangian evaluation implemented in AMPL using the following form

$$\nabla^2_{xx} L(x) = \nabla^2_{xx} \left[\sum_{i=0}^{n\_obj-1} OW[i] o_i(x) + \sigma \sum_{i=0}^{n\_con-1} Y[i] c_i(x) \right],$$

where o_i is the objective function, c_i is the constraint function, σ is a scaling factor commonly set to +1 or -1, $OW[i]$ is a scaling factor for objective function o_i, and $Y[i]$ is Lagrange multiplier for each c_i and equals zero when c_i is a free constraint. In our situation, we deal with a scalar objective function so $i = 1$ and $f = o_i$.

To call this routine, our program sets up the scale multiplier to be 1, $OW[0] = 1$, and the scale multiplier for the sum of constraints to be negative one, $\sigma = -1$. It copies the appropriate Lagrange multipliers to Y and calls the function `sphes`.

The result returns in the structure variable named `sputinfo` which is already in the compressed column vector format used by PATH. The matrix is stored as the top left corner of the MCP Jacobian matrix. The rest of the matrix is constructed using `jacval` and put it in an appropriate place in the MCP Jacobian matrix. Note that our program uses FORTRAN indices, which is a requirement for the PATH solver.

4.4 Results of the NLP solver

We assume that a user has created a nonlinear problem using the AMPL syntax and solves it by issuing the following commands:

```
option solver pathnlp;
solve;
```

Optionally, a user can guide the PATH solver using the option file, `path.opt` identified by

```
options pathnlp_options "optfile=path.opt";
```

Alternatively, a user can specify the options directly using the following syntax

```
options pathnlp_options "option_name=option_value";
```

Note that `option_name` must be a valid option of the PATH solver (see [25]). For example, to see the warning messages and current option settings of the PATH solver, a user can specify the following:

```
options pathnlp_options "output_warnings=yes output_options=yes";
```

To increase the number of iterations, a user can specify

```
options pathnlp_options
"major_iteration_limit=1000 minor_iteration_limit=10000";
```

To decrease the convergence tolerance from 1×10^{-6} to 1×10^{-8}, a user can specify

```
options pathnlp_options "convergence_tolerance=1E-8";
```

Consult [25, 24] for more details.

4.4.1 Results of the Hock/Schittkowski test suite.

We tested `pathnlp` with and without the new merit function using AMPL models of the Hock/Schittkowski test suite. This test used 115 NLP problems, two of which are incomplete. All problems are retrieved from the collection of AMPL test problems web site, `http://www.ampl.com/ampl`. This Hock/Schittkowski test suite was implemented by Professor Robert Vanderbei.

From the 113 NLP problems, 59 problems are unconstrained nonlinear programs, 48 problems have only equality constraints, while 3 problems contain range constraints and 3 problems have both equality and range constraints. We compare our results with four different NLP solvers available in AMPL, LANCELOT [13], MINOS [56], NPSOL [30] and SNOPT [29]. All solvers run using their default options. The PATH solver with the new merit function uses the weight $\gamma = 0.30$.

Table 12 shows details of these test runs on the Hock/Schittkowski test suite.

Table 12: Number of final solutions reported from 5 nonlinear solvers

Solver	`Fail`	`Infea`	`No prog`	`Iter`	`Local`	`Optimal`	`KKT`
LANCELOT	1	2	9	8	2	91	93
MINOS	0	1	0	7	11	94	105
NPSOL	7	0	2	0	8	96	104
PATH	0	0	10	0	21	82	103
PATH (merit)	0	0	5	0	20	88	108
SNOPT	0	0	2	12	4	95	99
Total	8	3	28	27	66	546	

Here `Fail` identifies the number of errors that occur because of an unexpected break from the solver, `Infea` identifies the number of solutions that are termed by the solver to be infeasible, `No prog` identifies the number of solutions that cannot be improved upon the current point by the solver, `Iter` identifies the number of times that the solver reached its default iteration limits, `Local` indicates the number of solutions that the solver found solutions that are different from reported global solutions, `Optimal` identifies the number of optimal solutions that are the same as reported optimal solutions, and `KKT` identifies the sum of `Local` and `Optimal`, which are KKT solutions.

The PATHNLP solver with the new merit function is very effective for solving this problem suite, solving 108 out of 113 problems. It is certainly comparable to the other NLP solvers listed here. Furthermore, the new merit function improves the robustness of the PATH code over the default version.

The test suite provides an indication of the global solution for each of the problems. Comparing these values to those found by our algorithms, the columns labeled `Local` and `Optimal` can be generated. As one can see from the local solution column, the PATHNLP solver is more likely to find first-order points that are not globally optimal for this given test problems. A more complete breakdown of the failures is given in Table 13.

Table 13: Number of nonoptimal solutions reported from 5 nonlinear solvers

Solver	`Unconstrained`	`Equalities`	`Ranges`	`Both`	`Total`
LANCELOT	19	3	0	0	22
MINOS	9	10	0	0	19
NPSOL	11	5	0	0	16
PATH	23	8	0	0	31
PATH (merit)	16	6	3	0	25
SNOPT	15	3	0	0	18
Total	59	48	3	3	

It is clear that for finding globally optimal solutions, the NPSOL solver is the most effective solver, failing only 16 times.

Table 14 reports the total timing of nonoptimal and optimal solutions from each solver in seconds. Results were tested on the 330 MHz Ultra Sparc machine with 64 MB RAM running SunOS version 5.6.

Table 14: Total timing of nonoptimal and optimal solutions from 5 nonlinear solvers

Solver	Nonoptimal	Optimal	Total
LANCELOT	127.52	123.24	250.76
MINOS	352.47	39.66	392.13
NPSOL	130.91	60.10	191.01
PATH	107.15	100.30	207.45
PATH (merit)	63.43	78.95	142.38
SNOPT	9.23	34.83	44.06
Total	790.71	437.08	1227.79

From table 14, SNOPT uses less time to solve this problem suite. It spends only 20.95% of the total time on nonoptimal solutions or failures. MINOS consumes the largest times to find nonoptimal solutions but is comparable to SNOPT for finding globally optimal solutions. Our PATHNLP solver with the merit function reduces the total time by 31.36% from the default version of PATH. Clearly, these problems are too small to derive many definitive conclusions on speed.

4.4.2 Results of large problem test suites

We select 4 problems from each of four different test suites, Markowitz Models for Portfolio Optimization, Minimal Surfaces, Nonnegative Least Squares and Structural optimization. All problems are retrieved from the collection of AMPL test problems web site, `http://www.ampl.com/ampl`. From Markowitz Models for Portfolio Optimization test suite, we select the `markowitz2.mod` which has 1200 variables and 201 equality constraints. From Minimal Surfaces test suite, we select the `minsurf.mod` which has 1681 variables and no constraints. From Nonnegative Least Squares test suite, we select the `nnls2.mod` which has 543 variables and 393 equality constraints. From Structural optimization test suite, we select the `structure8.mod` which has 13448 variables and 13448 equality constraints.

Table 15 summarizes the result of our test runs on large problem sets. Results were tested on Sparc machine with 245 MB RAM running SunOS version 5.5.1.

Table 15: Total timing from 5 nonlinear solvers

Solver	`Markowitz`	`Minimal`	`Nonnegative`	`Structural`
LANCELOT	503	106	3	*mem*
MINOS	*sup*	*sup*	*sup*	*inf*
NPSOL	538	657	191	*mem*
PATH	84	333	2	*res*
PATH (merit)	123	221	4	18,375
SNOPT	*itr*	*sup*	*sup*	*ini*

Here a keyword in the table identifies that the solver has difficulty solving this problem, where *mem* identifies that the solver could not allocate enough spaces, *sup* identifies that the solver reported the superbasics limit is too small, *itr* identifies that the solver reached its iteration limits, *inf* identifies that the solver reported problem is unbounded, *res* identifies that the solver exceeds the resource limits and *ini* identifies that the solver found the problem is infeasible due to a bad starting point. Solutions from all successfully solved problems are the same for all solvers. The solution for `markowitz2.mod` is -0.526165. The solution for `minsurf.mod` is 7.611023. The solution for `nnls2.mod` is 32.644706. The solution for `structure8.mod` is 1039.825620. Note that MINOS and SNOPT failed to solve any large problems, while PATHNLP with merit function solved all of them. This shows the efficiency of our code for handling large problem sets which is essential for solving real world models.

Chapter 5

Conclusion and future research

5.1 Summary of thesis work

We have exhibited a method for simulation optimization based on the sample-path technique using a gradient based optimization solver. This method is applicable for solving an engineering simulation design problem that incorporates decision variable constraints within a modeling language such as GAMS, commonly used in many areas of economics, business and engineering. Our thesis work also exhibits how modular implementations of different components such as the optimization solver, the external module, the gradient approximation module can be replaced by new technologies without effecting the other components of the solver. This shows the potential of applying the new state-of-the-art optimization solvers to deal with even more difficult problems.

In our simulation optimization method, we treat the simulation module as a black-box function which is implemented outside the modeling language system. Its implementation can be written in a standard programming language such as C, FORTRAN or JAVA or in a shell script that extracts data directly from the system. This opens the opportunity for researchers to make use of optimization algorithm without re-implementing the optimization module. They can concentrate on the validity and verification of the simulation model, the analysis of the optimization results and its interpretation. They can also add or modify the constraints to help the optimization algorithm to find suitable solutions for their real problems.

By using CONDOR, the algorithm makes use of all available computing resources within a network to solve large computational simulation optimizations. The implementation is transparent in that a practitioner does not have to understand anything about CONDOR. In fact, a practitioner can specify to use CONDOR via an option file from our solver implementation.

To use a Master-Worker server, a practitioner needs to start the server as a separate process before executing the GAMS model. The Master-Worker server terminates only when the practitioner changes the status file to contain the termination code. The major advantage of using the Master-Worker server over using a CONDOR submit file is a reduction in waiting time for an idle machines.

Typically, a researcher acquires one or more nonlinear optimization algorithms to find the solution of the problem because each algorithm exhibits different strengths and weaknesses to cope with a specific nonlinear problem and formulation. Our techniques show how to expand the use of the mixed complementarity solver to solve nonlinear programs. This expands the number of nonlinear solvers to cope with difficult nonlinear problems. A researcher can test the strength of each nonlinear solver to determine the best solution of the problem.

5.2 Future research

There are many research directions that can enhance the capability of the simulation optimization solver to solve large stochastic optimizations, or to expand the number of nonlinear programming solvers.

1. In this thesis, we deal with continuous decision variables because of the use of a gradient based optimization solver. In real simulation models, the mixture of discrete and continuous decision variables is common. A joint method for solving simulation

optimization could use gradient based optimization together with search algorithms such as the branch and bound method, genetic algorithms, nested partitions or tabu search.

2. We explored the use of the quadratic model to estimate the gradient. For a specific simulation function, this quadratic model may not be appropriate. The current implementation deals with this simulation by fitting the quadratic model with a very small radius. Future research could explore other function models such as a piecewise linear model, an exponential model, a polynomial of higher degree or combinations of the above. The derivative computations of a known model can be computed using automatic differentiation which will guarantee the result up to the current machine precision. However, this model must overcome the over-fitting problems that can easily capture a noise.

3. In this thesis, we link the simulation module by either writing a driver routine and compiling it together with our quadratic simulation module or making the system call to a separate executable simulation code. We could expand the possibility of using the simulation computation directly from a commercial simulation software such as ACSL (Advanced Continuous Simulation Language simulates continuous process such as the production of sound, the action of a drug, etc.), ATHENA (ATHENA is a chemical simulators), or Fluent (Fluid Flow Modeling and Analysis uses for advanced physical models for turbulence, combustion and multiphase applications.), etc.

4. In this thesis, we solve the single-stage deterministic simulation optimization problem. That means the simulation computation is not changed during the course of computation. However, we could solve the multi-stage deterministic simulation

optimization using our current implementation. The modification of the simulation module does not effect the optimization algorithm as long as the simulation is deterministic. The nondeterministic or nonparametric simulation optimization is the subject of further research.

5. From the nonlinear solver point of view, we can expand our research to hook up more mixed complementarity solvers and search for the best choice of merit functions.

Bibliography

[1] Allen, A. O. *Probability, Statistics and Queueing Theory*, second ed. Academic Press, London, 1990.

[2] Anbar, D. On optimal estimation methods using stochastic approximation procedures. *The Annals of Statistics 1* (1973), 1175–1184.

[3] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Cros, J. D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. *LAPACK User's Guide*, second ed. SIAM, Philadelphia, Pennsylvania, 1995.

[4] Attouch, H. *Variational Convergence for Functions and Operators.* Pitman, 1984.

[5] Azadivar, F., and Lee, Y. H. Optimization of discrete variable stochastic systems by computer simulation. *Mathematics and Computers in Simulation 30* (1988), 331–345.

[6] Azadivar, F., and Talavage, J. Optimization of stochastic simulation models. *Mathematics and Computation in Simulation 22* (1980), 231–241.

[7] Azadivar, F., and Tompkins, G. Genetic algorithms in optimizing simulated systems. *WSC95 1995 Winter Simulation Conference Proceedings* (1995), 757–762.

[8] Banks, J. Introduction to simulation. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 7–13.

[9] Barton, R. R. Minimization algorithms for functions with random noise. *American Journal of Mathematical and Management Sciences 4* (1984), 109–138.

[10] Billups, S. C. *Algorithms for Complementarity Problems and Generalized Equations.* PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, Aug. 1995.

[11] Bongartz, I., Conn, A. R., Gould, N., and Toint, P. L. CUTE: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software 21* (1995), 123–160.

[12] Brooke, A., Kendrick, D., and Meeraus, A. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.

[13] Conn, A. R., Gould, N. I. M., and Toint, P. L. *LANCELOT: A Fortran package for Large–Scale Nonlinear Optimization (Release A)*. No. 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, 1992.

[14] Conn, A. R., and Toint, P. L. An algorithm using quadratic interpolation for unconstrained derivative free optimization. In *Nonlinear Optimization and Applications*, G. D. Pillo and F. Giannessi, Eds. Plenum Press, New York, 1996.

[15] Cramér, H. *Mathematical Methods of Statistics*. Princeton University Press, N.J, 1946.

[16] de Mello, T. H. *Simulation-Based Methods for Stochastic Optimization*. PhD thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 1998.

[17] Dennis, J. E., and Torczon, V. Direct search methods on parallel machines. *SIAM Journal on Optimization 1* (1991), 448–474.

[18] Dirkse, S. P., and Ferris, M. C. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software 5* (1995), 123–156.

[19] Eaves, B. C. A short course in solving equations with PL homotopies. In *Nonlinear Programming* (Providence, RI, 1976), R. W. Cottle and C. E. Lemke, Eds., American Mathematical Society, SIAM–AMS Proceedings, pp. 73–143.

[20] El-Bakry, A. S., Tapia, R. A., Tsuchiya, T., and Zhang, Y. On the formulation and theory of the primal-dual Newton interior-point method for nonlinear programming. Technical Report TR92-40, Department of Computational and Applied Mathematics, Rice University, 1992.

[21] Epema, D. H. J., Livny, M., van Dantzig, R., Evers, X., and Pruyne, J. A worldwide flock of condors: Load sharing among workstation clusters. *Journal on Future Generations of Computer Systems 12* (1996), 53–65.

[22] Ferris, M. C., Fourer, R., and Gay, D. M. Expressing complementarity problems and communicating them to solvers. *SIAM Journal on Optimization 9* (1999), 991–1009.

[23] Ferris, M. C., Kanzow, C., and Munson, T. S. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming 86* (1999), 475–497.

[24] Ferris, M. C., and Munson, T. S. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications 12* (1999), 207–227.

[25] Ferris, M. C., and Munson, T. S. Complementarity problems in GAMS and the PATH solver. *Journal of Economic Dynamics and Control 24* (2000), 165–188.

[26] Ferris, M. C., Munson, T. S., and Sinapiromsaran, K. A practical approach to sample-path simulation optimization. Data Mining Institute Technical Report 00-03, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.

[27] Fischer, A. A special Newton–type optimization method. *Optimization 24* (1992), 269–284.

[28] Gay, D. M. Hooking your solver to AMPL. Tech. rep., Bell Laboratories, Murray Hill, New Jersey, 1997. Revised 1994, 1997.

[29] Gill, P. E., Murray, W., and Saunders, M. A. SNOPT: An SQP algorithm for large-scale constrained optimization. Report NA 97-2, Department of Mathematics, University of California, San Diego, San Diego, California, 1997.

[30] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H. User's Guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming. Technical Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, California, January 1986.

[31] Glasserman, P. *Gradient Estimation via Perturbation Analysis*. Kluwer, Norwell, MA, 1991.

[32] Glover, F., Kelly, J. P., and Laguna, M. New advances for wedding optimization and simulation. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 255–260.

[33] Glynn, P. W. Optimization of stochastic systems. *WSC86 1986 Winter Simulation Conference Proceedings* (1986), 52–59.

[34] Goux, J.-P., Linderoth, J., and Yoder, M. Metacomputing and the master-worker paradigm. Tech. rep., Argonne National Laboratory, 1999.

[35] Griewank, A., and Corliss, G. F., Eds. *Automatic Differentiation of Algorithms: Theory, Implementation and Application.* SIAM, Philadelphia, Pennsylvania, 1991.

[36] Griewank, A., Juedes, D., and Utke, J. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software 22* (1996), 131–167.

[37] Gürkan, G., Özge, A. Y., and Robinson, S. M. Sample-path solution of stochastic variational inequalities, with applications to option pricing. *WSC96 1996 Winter Simulation Conference Proceedings* (1996), 337–344.

[38] Gürkan, G., Özge, A. Y., and Robinson, S. M. Sample-path solution of stochastic variational inequalities. *Mathematical Programming 84* (1999), 313–333.

[39] Gürkan, G., Özge, A. Y., and Robinson, S. M. Solving stochastic optimization problems with stochastic constraints: An application in network design. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 471–478.

[40] Haddock, J., and Mittenthal, J. Simulation optimization using simulated annealing. *Comput. Ind. Eng. 22* (1992), 387–395.

[41] Hill, R. R. A monte carlo study of genetic algorithm initial population generation methods. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 543–547.

[42] Ho, Y. C., and Cao, X. R. *Perturbation analysis of discrete event dynamic systems.* Kluwer, 1991.

[43] Hock, W., and Schittkowski, K. *Test Examples for Nonlinear Programming Codes*, vol. 187 of *Lecture Notes in Economics and Mathematical Systems.* Springer Verlag, Berlin, Germany, 1981.

[44] Kall, P. Approximation to optimization problems: An elementary review. *Mathematics of Operations Research* (1986), 9–18.

[45] Karush, W. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, 1939.

[46] Keifer, J., and Wolfowitz, J. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics 23* (1952), 462–466.

[47] Kleijnen, P. C. Validation of models: Statistical techniques and data availability. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 647–654.

[48] Kleywegt, A., and Shapiro, A. The sample average approximation method for stochastic discrete optimization. *Stochastic Programming E-Print Series* (1999). Available from dochost.rz.hu-berlin.de/speps.

[49] Kuhn, H. W., and Tucker, A. W. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, J. Neyman, Ed. University of California Press, Berkeley and Los Angeles, 1951, pp. 481–492.

[50] Law, A. M., and McComas, M. G. Simulation of manufacturing systems. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 56–59.

[51] Lewis, R., and Torczon, V. Pattern search methods for linearly constrained minimization. *SIAM Journal on Optimization 10* (2000), 917–941.

[52] Litzkow, M., Tannenbaum, T., Basney, J., and Livny, M. Checkpoint and migration of unix processes in the condor distributed processing system. Tech. Rep. 1346, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1997. http://www.cs.wisc.edu/condor/doc/ckpt97.ps.

[53] Litzkow, M. J., Livny, M., and Mutka, M. W. Condor: A hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems* (June 1988), pp. 104–111.

[54] Livny, M. PI, the Condor project, high throughput computing. www.cs.wisc.edu/condor.

[55] Morito, S., Koida, J., Iwama, T., Sato, M., and Tamura, Y. Simulation-based constraint generation with applications to optimization of logistic system design. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 531–536.

[56] Murtagh, B. A., and Saunders, M. A. MINOS 5.0 user's guide. Technical Report SOL 83.20, Stanford University, Stanford, California, 1983.

[57] Nath, S. K., Escobedo, F. A., and de Pablo, J. J. On the simulation of vapor-liquid equilibria for alkanes. *Journal of Chemical Physics 108* (1998), 9905–9911.

[58] Ólafsson, S., and Shi, L. Optimization via adaptive sampling and regenerative simulation. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 666–672.

[59] Page, M., Gensel, J., and Boudis, M. An algorithm for goal-driven simulation. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 578–585.

[60] Plambeck, E. L., Fu, B. R., Robinson, S., and Suri, R. Throughput optimization in tandem production lines via nonsmooth programming. In *Proceedings of the 1993 Summer Computer Simulation Conference* (San Diego, CA, 1993), J. Schoen, Ed., Society for Computer Simulation, pp. 70–75.

[61] Plambeck, E. L., Fu, B. R., Robinson, S., and Suri, R. Sample-path optimization of convex stochastic performance functions. *Mathematical Programming 75* (1996), 137–176.

[62] Ralph, D. Global convergence of damped Newton's method for nonsmooth equations, via the path search. *Mathematics of Operations Research 19* (1994), 352–389.

[63] Robbins, H., and Monro, S. A stochastic approximation method. *Annals of Mathematical Statistics 22* (1951), 400–407.

[64] Robinson, S. M. Normal maps induced by linear transformations. *Mathematics of Operations Research 17* (1992), 691–714.

[65] Robinson, S. M. Newton's method for a class of nonsmooth functions. *Set Valued Analysis 2* (1994), 291–305.

[66] Robinson, S. M. Analysis of sample-path optimization. *Mathematics of Operations Research 21* (1996), 513–528.

[67] Rockafellar, R. T. *Convex Analysis.* Princeton University Press, Princeton, New Jersey, 1970.

[68] Rubinstein, R. Y., and Shapiro, A. Optimization of static simulation models by the scroe function method. *Mathematics and Computers in Simulation 32* (1990), 373–392.

[69] Rubinstein, R. Y., and Shapiro, A. *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*. Wiley, Chichester, 1993.

[70] Sadowski, D. A., and Grabau, M. R. Tips for successful practice of simulation. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 60–66.

[71] Santos, M. I., and Nova, M. O. P. The main issues in nonlinear simulation metamodel estimation. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 502–509.

[72] Sargent, R. G. Validation and verification of simulation models. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 39–48.

[73] Segreti, A. C., Carter, W. H., and Wampler, G. L. Monte carlo evaluation of several sequential optimization techniques when the response is time to an event. *Journal of Statistical Computation and Simulation 9* (1979), 289–301.

[74] Shairo, A., and Homem-del-Mello, T. A simulation-based approach to two-stage stochastic programming with recourse. *Mathematical Programming 81* (1998), 301–325.

[75] Shannon, R. E. Introduction to the art and science of simulation. *WSC98 1998 Winter simulation conference proceedings 1* (1998), 7–14.

[76] Shi, L. An integrated framework for determinstic and stochastic optimization. *WSC97 1997 Winter Simulation Conference Proceedings* (1997), 358–365.

[77] Shi, L. Stopping criterion for a simulation-based optimization method. *WSC98 1998 Winter Simulation Conference Proceedings* (1998), 743–750.

[78] Smith, D. E. An empirical investigation of optimum seeking in the computer simulation situation. *Operations Research 21* (1973), 475–497.

[79] Spall, J. C. Stochastic optimization and the simulultaneous perturbation method. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 101–109.

[80] STANDRIDGE, C. R. A tutorial on simulation in health care: Applications and issues. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 49–55.

[81] STEPHENS, M. A. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association 69* (1974), 730–737.

[82] STEPHENS, M. A. Asymptotic results for goodness-of-fit statistics with unknown parameters. *Annals of Statistics 4* (1976), 357–369.

[83] TORCZON, V. On the convergence of the multidirectional search algorithm. *SIAM Journal on Optimization 1* (1991), 123–145.

[84] TORCZON, V. On the convergence of pattern search algorithms. *SIAM Journal on Optimization 7* (1997), 1–25.

[85] W. C. HEWITT, J., AND MILLER, E. E. Applying simulation in a consulting environment – tips from airport planners. *WSC99 1999 Winter Simulation Conference Proceedings* (1999), 67–71.

Appendix A

Simulation optimization GAMS model

A.1 M/M/1 queueing model

This is the complete GAMS model for the M/M/1 queue. The model sets up variables, parameters and equations using the GAMS syntax. It writes the simulation option file containing the dimension of the decision variables both dependent and independent, the name of the simulation configuration file, the number of sampling points, the radius adjustment and other configurable options. It also creates the configuration file for the quadratic simulation optimization module containing the number of server, the dimension of the output, the arrival rate of the queue, the number of simulation length, the random number seed, and the lower and upper bound of the service rate.

```
$Title M/M/1
$Offupper

Variables obj 'The objective variable to minimize',
          mu  'The simulation variable, service rate',
          w   'The simulation output, average waiting time';

Scalars n      'Number of independent variable'
        m      'Number of output parameter from simulation function'
        ns     'Exact number of simulation points needed for Quadratic'
        np     'Number of sampling points in each run'
        nw     'Additional sampling points in a run'
        tn     'Total number of parameters passing to CONOPTX'
        c      'constant'
        lambda 'Inter-arrival rate of customers'
        ransim 'Current random number'
```

```
        cusd   'Number of customers to be dropped from simulation'
        cusn   'Number of customers to be used in simulation'
count  'Count the number of locally optimal values'
        diffv  'Different between avgsol and aptsol';
m  = 1;
n  = 1;
ns = n*(n + 1)/2 + n + 1;
np = round(ns + ns/3);
nw = 10-np;
tn = n + m;

c = 4.0;
lambda = 3.0;
ransim = 578913;
cusd = 10000;
cusn = 1000000;

Sets iter /1*100/
     val  /'mu', 'wait', 'obj', 'ran'/;

Parameters maxsol(val) 'Maximum among iter runs'
           minsol(val) 'Minimum among iter runs'
           avgsol(val) 'Expected value among iter runs'
           aptsol(val) 'Solution that have objective value close to avgsol'
           allsol(val, iter) 'All solutions'
           stasol(iter) 'Status of each solution';

Equations cost    'The cost objective function',
          extcall 'External function call to CONOPTX';

cost..
    obj =e= sqr(mu-c) + w;

* the following equation is w - f(mu) = 0, defined externally
extcall..
    1*mu + 2*w =x= 1;

* Set the lower bound constraint on service rate
mu.lo = lambda;

* Construct the model and solve
Model mm1 /all/;
```

```
* Setting for GAMS
option decimals = 8;
option seed=1001;

* Select CONOPTX as the solver
option nlp = conoptx;

* Declare for qso.in file
file qsoin /qso.in/;
qsoin.ap = 0;
qsoin.nj = 2;

* Declare for qso.cfn file
file qsocfn /qso.cfn/;
qsocfn.ap = 0;
qsocfn.nj = 2;

* Initialize mu, w, obj
mu.l = 0.0;
w.l = 0.0;
obj.l = 0.0;

* Write the qso.in
put qsoin;
put 'SOIND=' n:1:0 /;
put 'SODEP=' m:1:0 /;
put 'SOTIN=' tn:1:0 /;
put 'SO_HI=1.0D+13' /;
put 'QS_NP=' np:2:0 /;
put 'QS_NW=' nw:2:0 /;
put 'SGCFN=qso.cfn' /;
put 'SGSFN=simple.sta' /;
put 'SOLFN=qso.log' /;
put 'QS_SR=1.0' /;
put 'QS_R2=0.999999' /;
put 'QS_RX=0.00000001' /;
put 'QS_MT=40' /;
put 'QS_UR=0.2' /;
put 'QS_AL=0.01' /;
put 'QS_FG=3' /;
put 'SO_FG=0' /;
put 'SG_FG=0' /;
putclose qsoin;
```

```
* Write the qso.cfn for simulation generator configuration
put qsocfn;
put n:1:0 /;
put m:1:0 /;
put lambda:5:2 /;
put '1000' /;
put '1000.0' /;
put ransim:9:0 /;
put cusd:10:0 /;
put cusn:20:0 /;
put mu.lo:5:0 /;
putclose qsocfn;

option iterlim = 100000;
option reslim = 100000;
mm1.optfile = 1;
Solve mm1 using nlp minimizing obj;
```

A.2 Telemarketing model

The GAMS model for the telemarketing simulation queue with 4 servers and one waiting queue is provided below. The main process is similar to the M/M/1 model. It differs in the modeling details and the configuration file. The configuration file contains the random number seed, the number of customers, the number of servers, the maximum queue length, the arrival rate, the probability that the customer will call back and the average waiting time before the customer call back, and the lower or upper bound of decision variables.

```
$Title Simulation Optimization:The call-waiting queue with 4 servers
$Offupper

* runno is used for generate the ".sta"
$setglobal runno 0

$Ontext
   The call-waiting queue composes of 4 servers and one waiting queue.
```

```
The customer calls this queue. The following situation occurs
1. Some servers are idle, then s/he will be served directly.
2. All servers are busy and the queue is not full, then s/he will
   decide either to wait in the queue with probability p or leaving
   the system with probability 1-p
3. All servers are busy and the queue is full, then s/he will leave
   the system.
Customers are served using FCFS service discipline. All servers are
being selected using the Round Robin order. The length of queue
is limited by M. Assuming the inter-arrival has the exponential
distribution with rate lambda.

The queue collects the 'average waiting time' for a typical customer and
the 'percentage of customers lost' for the system.
$Offtext

Scalars MaxQ    'Maximum number of customers in the queue'
        p       'Probability of customer call back without being served'
        cbwait  'Waiting time before a customer call back'
        lambda  'Inter-arrival rate'
        cusn    'Number of customers to use in a simulation'
        seedran 'Random number seed for simulation generator'
        n       'Number of independent variables'
m       'Output parameter from CONOPTX'
ns      'Exact number of simulation points to fit quadratic'
np      'Number of sampling simulation points for each run'
nw      'Additional sampling in one run'
tn      'Number of parameter passing to CONOPTX'
;
MaxQ   = 100;
p      = 0.1;
cbwait = 0.4;
lambda = 3.0;
cusn   = 10000;
* Generate a random number for Simulation generator
seedran = 458239;


Set i /1*4/;

* Set up for configuration files
m  = 2;
n  = card(i);
```

```
ns = n*(n+1)/2 + n + 1;
np = round(ns + ns/3);
*nw = 2*ns - np;
nw = 15;
tn = n + m;

Parameter r(i)  'Rate of improving server i';
r(i) = 5*(ord(i) - 1);
r('1') = 1;

Variables obj   'Objective function'
  w     'Average waiting time'
  plost 'Percentage of customers lost'
  sizeq 'the total queue size'
  x(i)  'Service rate for each servers';

* Set bounds
sizeq.fx = MaxQ;
*plost.lo = 0.0;
x.lo(i) = 0.5;
x.up(i) = 10.0;
*w.lo = 0.0;
*plost.up = 0.35;

* Guess the starting points
*x.l(i) = 1.0;

Equations COST    'Objective cost function'
          EXPLOST 'Expected percentage of customers lost'
          EXPWAIT 'Expected waiting time for a typical customer'
;

COST..
  obj =e= sum(i, r(i)*(x(i) - x.lo(i))) + 100*plost;

* the following equation is plost - f(1) = 0, defined externally
EXPLOST..
  sum(i, ord(i)*x(i)) + 5*plost + 6*w =x= 1;

* the following equation is w - f(2) = 0, defined externally
EXPWAIT..
  sum(i, ord(i)*x(i)) + 5*plost + 6*w =x= 2;
```

```
Model callqueue /all/;

** File declaration
file qsoin /qso.in/;
qsoin.nj = 2;
file qsocfn /qso.cfn/;
qsocfn.nj = 2;

* Write the qso.in
put qsoin;
put 'SOIND=' n:20:0 /;
put 'SODEP=' m:20:0 /;
put 'SOTIN=' tn:20:0 /;
put 'QS_NP=' np:20:0 /;
put 'QS_NW=' nw:20:0 /;
put 'SGCFN=qso.cfn' /;
put 'SGSFN=simple%runno%.sta' /;
put 'SOLFN=qso.log' /;
put 'QS_SR=4.0' /;
put 'QS_RX=0.00000001' /;
put 'QS_R2=0.9999999' /;
put 'QS_MT=40' /;
put 'QS_UR=0.5' /;
put 'QS_FG=3' /;
put 'SO_FG=0' /;
put 'SG_FG=14' /;
putclose qsoin;

* Write the qso.cfn for Simulation generator configuration
put qsocfn;
put seedran:9:0 /;
put cusn:20:0 /;
put card(i):2:0 /;
put MaxQ:10:0 /;
put lambda:10:8 /;
put p:10:8 /;
put cbwait:10:8 /;
loop(i, put '0.5 '); put /;
putclose qsocfn;

* Set options
option nlp = conoptx;
option iterlim = 100000;
```

```
option reslim = 100000;
option decimals = 8;
callqueue.optfile=1;

Solve callqueue using nlp minimizing obj;
display x.l, plost.l, w.l, obj.l;
```

A.3 Tandem production line model

The following is the tandem production line GAMS model of five machines with the sum of machine rates of the first 3 machines equal to 0.966 and the rest equal to 0.6. The main process is also similar to the M/M/1 model. The configuration file contains the number of machines, the number of runs, the simulation length, the time to start trace (for debugging purpose), the time to start the trace graph (for debugging purpose), the machine initial status, the flow rates of each machine, the machine failure rates, the machine repair rates, the buffer sizes, the random number seeds and the lower and upper bound of the flow rates.

```
$Title Tandem production line (6a): with five machines
$Offupper

* runno is used for generate the ".sta"
$setglobal runno 10

Set i 'Set of all machines' /1*5/;
Alias (i, j);

Variables flwr(i) 'Flow rates for each machine, i'
          tpinv   'Output simulation = Inverse throughput'
          obj     'Objective variable';

* Set the lower bound and upper bound
* If flow rates is zero, the production line will not move.
flwr.lo(i) = 0.05;

* Set the starting flow rates
flwr.l('1') = 0.800;
```

```
flwr.l('2') = 0.060;
flwr.l('3') = 0.106;
flwr.l('4') = 0.070;
flwr.l('5') = 0.530;
tpinv.l = 0.0;
obj.l = 0.0;

* Compute the last index for using to return tpinv
Scalars n  'Number of independent variables'
        m  'Output parameter from CONOPTX'
        ns 'Exact number of simulation points to fit quadratic'
        np 'Number of sampling simulation points for each run'
        nw 'Additional sampling in one run'
        tn 'Number of parameter passing to CONOPTX';
m  = 1;
n  = card(i);
ns = n*(n+1)/2 + n + 1;
np = round(ns + ns/3);
nw = 2*ns - np;
tn = n + m;

Scalars random1, random2;

Equation objective 'Objective function'
 flow1     'Flow rate constraint'
 flow2     'Flow rate constraint'
 simcon    'Simulation constraint';

objective..
  obj =e= tpinv;

* the following equation is w - f(mu) = 0, defined externally
simcon..
  sum(i, ord(i)*flwr(i)) + (n+m)*tpinv =x= 1;

flow1..
   sum(i$(ord(i) < 4), flwr(i)) =e= 0.966;

flow2..
   sum(i$(ord(i) > 3), flwr(i)) =e= 0.6;


* Create a model using all information
```

```
model simple /all/;

* Pick CONOPTX as the nonlinear programming solver.
option nlp = conoptx;
option Iterlim=10000;
option reslim=10000;
option random1:0;
option random2:0;
option decimals=8;

* File declaration
file qsoin /qso.in/;
qsoin.nj = 2;
file qsocfn /qso.cfn/;
qsocfn.nj = 2;

* Write the qso.in
put qsoin;
put 'SOIND=' n:20:0 /;
put 'SODEP=' m:20:0 /;
put 'SOTIN=' tn:20:0 /;
put 'QS_NP=' np:20:0 /;
put 'QS_NW=' nw:20:0 /;
put 'SGCFN=qso.cfn' /;
put 'SGSFN=simple%runno%.sta' /;
put 'SOLFN=qso.log' /;
put 'QS_SR=' n:5:1 /;
put 'QS_R2=0.99999' /;
put 'QS_RX=0.00000001' /;
put 'QS_MT=40' /;
put 'QS_UR=0.5' /;
put 'QS_FG=3' /;
put 'SO_FG=0' /;
put 'SG_FG=14' /;
putclose qsoin;

* Generate a random number for Simulation generator
random1 = 229780750;
random2 = 1460288881;

* Write the qso.cfn for Simulation generator configuration
put qsocfn;
put  n:20:0 /;
```

```
put '1         N             Number of runs(batches)' /;
put '4 500 49500' /;
put '0         t_trace' /;
put '0         t_graph' /;
loop(i, put '1 '); put /;
put 'FLWR '; loop(i, put flwr.l(i):8:3 ' ';); put /;
put 'MVTF 100 90 100 90 90' /;
put 'MTTR 10 4.5 6 4.5 4.5' /;
put 'BUFF 10 10 10 10' /;
put random1:10:0 ' ' random2:10:0 /;
loop(i, put '0.00001 '); put /;
putclose qsocfn;

* The solve statement
*simple.optfile=1;
solve simple using nlp minimizing obj;
display random1, random2, flwr.l, tpinv.l, obj.l;
```

Appendix B

CONDOR files and commands

B.1 Multiple architecture submit file

In order to use the available heterogeneous machines and checkpoint feature in the CONDOR environment, the user must compile binary executable codes that run on different computer architectures and operating systems. The CONDOR pool at the University of Wisconsin-Madison supports the following machine architecture/operating system configurations: ALPHA/OSF1, INTEL/LINUX, INTEL/SOLARIS26, INTEL/SOLARIS27, INTEL/WINNT40, SGI/IRIX6, SUN4u/SOLARIS26, SUN4u/SOLARIS27, SUN4x/SOLARIS26 and SUN4x/SOLARIS27.

The following CONDOR submit file makes use of three configuration types: the SUN4u/SOLARIS26, the INTEL/SOLARIS26 and the INTEL/LINUX.

```
NUMRUNS = 19
Initialdir = /p/gams/remote_condor/standalone/condor/unix/tandem/run
universe = standard
requirements = ((ARCH == "SUN4u" && OPSYS == "SOLARIS26") \
             || (ARCH == "INTEL" && OPSYS == "SOLARIS26") \
             || (ARCH == "INTEL" && OPSYS == "LINUX"))
executable = qso_$$(ARCH)_$$(OPSYS).exe
arguments = conqso_$(Process).ifn conqso_$(Process).ofn
rank=KFLOPS
log = condor_qso.log
notification = Error
queue $(NUMRUNS)
```

The user must supply three binary executable codes as qso_SUN4u_SOLARIS26.exe, qso_INTEL_SOLARIS26.exe and qso_INTEL_LINUX.exe in the initialdir.

B.2 CONDOR related commands

The following CONDOR commands are used to investigate and manage tasks within the CONDOR pool.

```
Usage: condor_q [options]
where [options] are
       -global                  Get global queue
       -submitter <submitter>   Get queue of specific submitter
       -help                    This screen
       -name <name>             Name of schedd
       -pool <host>             Use host as the central manager to query
       -long                    Verbose output
       -format <fmt> <attr>     Print attribute attr using format fmt
       -analyze                 Perform schedulability analysis on jobs
       -run                     Get information about running jobs
       -goodput                 Display job goodput statistics
       -cputime                 Display CPU_TIME instead of RUN_TIME
       -currentrun              Display times only for current run
       -io                      Show information regarding I/O
       restriction list
where each restriction may be one of
       <cluster>                Get information about specific cluster
       <cluster>.<proc>         Get information about specific job
       <owner>                  Information about jobs owned by <owner>
       -constraint <expr>       Add constraint on classads

Usage: condor_reschedule [general-options] [targets]
where [general-options] can be zero or more of:
       -help                gives this usage information
       -version             prints the version
       -pool hostname       use the given central manager to find daemons
where [targets] can be zero or more of:
       hostname             given host
       <ip.address:port>    given "sinful string"
       (for compatibility with other Condor tools, you can also use:)
       -name hostname       given host
       -addr <addr:port>    given "sinful string"
       (if no targets are specified, the local host is used)

condor_reschedule causes the condor_schedd to update the central manager
and initiate a new negotiation cycle.
```

```
Usage: condor_rm [options] [constraints]
where [options] is zero or more of:
       -help              Display this message and exit
       -version           Display version information and exit
       -name schedd_name  Connect to the given schedd
       -pool hostname     Use the given central manager to find daemons
       -addr <ip:port>    Connect directly to the given "sinful string"
and where [constraints] is one or more of:
       cluster.proc       Removes the given job
       cluster            Removes the given cluster of jobs
       user               Removes all jobs owned by user
       -all               Removes all jobs

Usage: condor_submit [options] [cmdfile]
       Valid options:
       -v             verbose output
       -n schedd_name  submit to the specified schedd
       -r schedd_name  submit to the specified remote schedd
       -d             disable file permission checks

       If [cmdfile] is omitted, input is read from stdin
```

Appendix C

Options for the quadratic simulation optimization

C.1 General options

The user can guide the quadratic simulation optimization using the QSO option file which is divided into 3 sets of options. The first set of options deals with the simulation module. The second set of options deals with the simulation optimization in general. The third set of options deals with the quadratic approximation model and the CONDOR environment. These options are included in a file named 'qso.in' and their names must start at the beginning of the line.

```
_____=#####
NAME  VALUE
```

The option name composes of 5 characters and its value starts from column 7. The order of options are irrelevant. If the same option appears more than one line, the last option line will be used. A character in the sixth column will be ignored.

C.2 Simulation module options

These options deal with setting up the simulation file names and the output report of the simulation run.

```
SGCFN: Simulation generator customize file name, sgcfn
       The file name of the customizable options for the simulation
       generator, it will be called in sg_read for setting up the
       simulation generator.
```

```
default = 'qso.cfn'

SGEXE: Simulation generator executable command, sgexe
       The command line to execute the simulation generator if only
       the executable version is available.
default = 'qso.exe'

SGIFN: Simulation generator input file name, sgifn
       The input file name for simulation generator to read during the
       normal simulation run.
default = 'qso.ifn'

SGOFN: Simulation generator output file name, sgofn
       The output file name from the simulation generator after it
       simulates the result using the SGIFN as input and SGEXE as
       executable.
default = 'qso.ofn'

SGOPT: Simulation generator option file, sgopt
       The option file name for simulation generator
default = 'qso.opt'

SGSFN: Simulation generator statistical output file, sgsfn
       The output file name to report the statistics of simulation
       run, number of function and derivative calls.
default = 'qso.sta'

SG_FG: Simulation generator flag, sgflag
       Flag for report the information from the simulation generator
       using the bit pattern. So the user can set up any combinations of
       report.
       Here is the code with respect to the location of bit $2^k$ on
       1    Display SG_FG and QS_FG
       2    Display SGIFN, SGOFN, SGEXE, SGCFN, SGOPT, SG_DN, SG_CN
       3    Display seed, nw, np, relax, rsquare, radius, urad
    pfit, skew, nf, maxit, alpha, W2 threshold
       4    Display Simulation iterations
       SG_FG = 0 means no report.
default = 15                                        [positive integer]
```

C.3 Simulation optimization options

These options deal with setting up the simulation optimization in general and do not include the quadratic approximation, which is discussed in C.4.

```
MAXDZ: Size of DPARS (global setting for double precision parameters)
default = 20                                          [positive integer]

MAXIZ: Size of IPARS (global setting for integer parameters)
default = 40                                          [positive integer]

MAXMN: Maximum size of double precision for keeping x of size n and f(x)
       of size m
default = 53                                           [= MAX_N + MAX_M]

MAXNP: Maximum number of kept points.
default = 10000                                     [setting at 50*MAX_P]

MAXSZ: Maximum number of elements in A matrix construct from x during
       the least square fitted of the quadratic
default = 1326                     [= MAX_N (MAX_N + 1)/2 + MAX_N + 1]

MAX_I: Maximum number of iterations before giving one simulation run
default = 20

MAX_M: Maximum number of dependent variables
default = 3                                           [positive integer]

MAX_N: Maximum number of independent variables
default = 50                                          [positive integer]

MAX_P: Maximum number of sampling points at one time
default = 200                                         [positive integer]

MAX_S: Maximum number of loop delay before checking the output of
       simulation run. (only used when simulation source code is not
       available.)
default = 1000                                        [positive integer]

MAX_W: Maximum number of tries before giving up all simulation runs.
       (only used when simulation source code is not available.)
default = 100000                                      [positive integer]
```

```
SODEP: Simulation Optimization number of dependent variables, m
       The number of dependent variables for the simulation generator.
default = 1                                          [positive integer]

SOEPS: Simulation Optimization Epsilon, eps
       The largest number that is indistinguishable when adding to 1.0.
default = 3.667D-11                                          [eps > 0]

SOIND: Simulation Optimization number of independent variables, n
       The number of independent variables in the simulation, for M/M/1
       it is 1.
default = 1                                          [positive integer]

SOLFN: Simulation Optimization log file
       The name of the simulation optimization log file
default = 'qso.log'

SO_FG: Simulation Optimization general flag, soflag
       Printing flag for the Quadratic simulation optimization
       We use the bit patterns, so the user can set up any option
       combinations.
       Here is the code with respect to the location of bit $2^k$ on
       1    Display SODEP,SOIND,SOTIN,SOEPS,SO_NI,SO_PI
       2    Display Hard lower bound and upper bound
       3    Display Return simulation function
       4    Display function and derivative calls
       5    Display Number of fits, current rsquare,
    free space and radius for each minor iteration
       6    Display W2 acceptance or rejection
       7    Display Skewness coefficient for each minor iteration
    and Histogram cutting
       8    Display Quadratic information
       SO_FG = 0 means no report.
default = 511                                        [positive integer]


SO_HI: Simulation Optimization general upper bound, upper
       The largest upper bound for x
default =  1.0D+20                                         [upper > 0]

SO_LO: Simulation Optimization general lower bound, lower
```

```
       The smallest lower bound for x
default = -1.0D+20                                           [lower < 0]

SO_NI: Simulation Optimization Negative infinity, neginf
       The smallest negative number to be used in the simulation
default = -1.0D+20                                          [neginf < 0]

SO_PI: Simulation Optimization Positive infinity, posinf
       The largest positive number to be used in the simulation
default = 1.0D+20                                           [posinf > 0]
```

C.4 Quadratic Simulation optimization options

These options deal with setting up the quadratic model for the gradient approximation and CONDOR environment.

```
CON_U: Condor universe flag, conuflag
       Flag for submitting the simulation via CONDOR
       1    Use the vanilla universe
       2    Use the standard universe
default = 1                                           [positive integer]

CON_S: Condor serial flag, consflag
       Flag for computing serial version while waiting
       Here is the code with respect to the location of bit $2^k$ on
       1    Apply serial computation during the waiting period
default = 1                                           [positive integer]

QS_AL: Quadratic Simulation significant level
       Level of signification testing for error having normal
       distribution with unknown mean and standard deviation.
default = 0.05        [Allowable value are 0.15, 0.1, 0.05, 0.025, 0.01]


QS_DR: Quadratic Simulation percentage of sampling region, delta
       Percentage of radius to use as the sampling region.
default = 100.0                                  [0.0 <= QS_DR <= 100.0]

QS_FG: Quadratic Simulation flag, qsflag
       Selecting the method of determine function and derivative calls.
       Here is the code with respect to the location of bit $2^k$ on
```

```
      1    Use delta
      2    Use adjusted R-square
      3    Use Cramer-von Mises Statistic or W^2 statistic
      QS_FG = 0 means no report.
default = 31                                      [positive integer]

QS_FI: Quadratic Simulation frequency interval, nf
       Number of interval to use for droping outline or unfit points
default = 20                                      [positive integer]

QS_MR: Quadratic Simulation minimum sampling radius, minrx
       Minimum radius for fitting QP
default = 0.0000001                                    [minrx > 0.0]

QS_MT: Quadratic Simulation minor iterations, maxit
       Maximum number of minor iterations during the radius reduction
       phase in the QP fit.
default = 30                                      [positive integer]

QS_NP: Quadratic Simulation number of sampling points, np
       Minimum requirement for the number of sampling points to fit QP
default = 50                          [ = n (n + 1)/2 + n + 1 + 1 ]

QS_NW: Quadratic Simulation number of additional sampling points, nw
       Additional sampling points to be called for the minor iteration.
default = 5                                     [setting as np / 10]

QS_PF: Quadratic Simulation percentage of fit, pfit
       Acceptable percentage of point to accept the QP.
default = 100.0                      [suggest 90.0 <= QS_PF <= 100.0]

QS_R2: Quadratic Simulation R^2 threshold, rsquare
       The R^2 threshold is used when the computed R^2 is greater than
       this threshold, then the QP is considering as fit.
       If the adjusted R^2 is not used,
       then it is fixed at 0.9999999 and cannot be changed.
default = 0.99                       [suggest 0.99 <= rsquare <= 1.0]

QS_RX: Quadratic Simulation relax distance, relax
       Maximum distance for considering two points to be the same using
       1-norm.
default = 0.00001                                      [relax > 0.0]
```

```
QS_SD: Quadratic Simulation random seed, seed
       Start random seed for the random number generator which is used
       internally for Quadratic Simulation Optimization for generating
       new sampling point, etc.
default = 458239                                      [positive integer]

QS_SR: Quadratic Simulation sampling radius, radius
       Starting radius for the first call to a point during the
       computation of derivative.
default = 1.0                                            [radius > 0.0]

QS_SW: Quadratic Simulation skewness threshold, skew
       The coefficient of skewness threshold for removing outliers,
       if the absolute of computed skewness is greater than skew then
       the Quadratic Simulation Optimization will try to remove outliers.
       Otherwise, it uses QS_UR to reduce the radius
default = 1.0                                               [skew >= 0]

QS_UR: Quadratic Simulation updated radius, urad
       Percentages of radius increasing after W^2 test successful or
       decreasing after W^2 test is unsatisfactory or the regular
       reduction of radius.
default = 0.4                                       [0 <= urad <= 1.0]
```