5-2011

Automated, Parallel Optimization Algorithms for Stochastic Functions

Dheeraj Chahal
Clemson University, dchahal@clemson.edu

Automated, Parallel Optimization Algorithms for Stochastic Functions

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

---

by
Dheeraj Chahal
May 2011

Accepted by:
Dr. Sebastian Goasguen, Committee Chair
Dr. Steve Stuart
Dr. D. E. Stevenson
Dr. Pradip Srimani

Abstract

The optimization algorithms for stochastic functions are desired specifically for real-world and simulation applications where results are obtained from sampling, and contain experimental error or random noise. We have developed a series of stochastic optimization algorithms based on the well-known classical down hill simplex algorithm. Our parallel implementation of these optimization algorithms, using a framework called MW, is based on a master-worker architecture where each worker runs a massively parallel program. This parallel implementation allows the sampling to proceed independently on many processors as demonstrated by scaling up to more than 100 vertices and 300 cores. This framework is highly suitable for clusters with an ever increasing number of cores per node. The new algorithms have been successfully applied to the reparameterization of a model for liquid water, achieving thermodynamic and structural results for liquid water that are better than a standard model used in molecular simulations, with the the advantage of a fully automated parameterization process.

Dedication

I dedicate my dissertation work to my teachers, my family and many friends.

Acknowledgments

I am heartily thankful to my advisor, Dr. Sebastian Goasguen, for his encouragent, support and guidance during this work. I am indebted to Dr. Steve Stuart for guiding me through this research. I greatfully acknowledge him and deeply appreciate his concern for my problems during my research. I admit that without his support this work could not be completed.

The members of my dissertation committee, Dr. D. E. Stevenson, and Dr. Pradip Srimani, have generously given their time and expertise to better my work. I thank them for their contribution and their support.

My wife, Dr. Neetu Tomar, has always been an inspiration for me. Her encouragement to pursue higher studies has resulted in this work.

All through the difficult phases during this work my daughter Srinidhi Chahal was a source of joy for me. I will always regret for denying her share of love for the first two years of my study at Clemson.

My parents, brother, sister-in-law, parents-in-law, brother-in-law should be acknowledged for motivating me for higher studies.

Special thanks to my friends Dr. Neeraj Gohad, Radheyshyam, Dr. Indranil Mitra, Rooplekha Mitra for their support and encouragement during the difficult times. I also acknowledge all my friends in India who were very supportive, though thousands of miles away.

I am also thankful to the School of Computing and Clemson University for providing the resources.

I would like to sincerely thank the whole hpc team for the timely responses and tireless efforts. I also thank physics department for providing the financial support in the form of teaching assistantship. In particular, I thank Mr. Jerry Hester for providing me an opportunity to work, teach and learn in the physics department.

My sincere thanks to Stuart research group in chemistry. They showed immense patience while listening to my research updates specially during group meetings. I hope, I compensated their fruitless effort to teach me chemistry by giving them my share of bagles.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In scientific and engineering research, parameter estimation and model calibration are common problems and requires rigorous simulations to understand the behavior of real systems. There is an increasing demand to design software to perform automatic optimization of some derived quantity known as objective function, a measure of "fitness" of the design. The computation of this objective function value is often an extremely computationally intensive process which must be performed numerous times to find the optimum parameters. An ability to deploy the parallel and distributed computing resources that form the basis of contemporary high performance computing architectures would be a distinct advantage in making automatic optimization a practical tool in the engineering design process [3] [4].

The best set of parameters which minimizes a cost function can be obtained by a number of optimization techniques. Thses optimization methods are broadly categorized as local optimzation methods and global optimization methods. The local optimization methods use the information from the neighborhood of the current approximation and always converge to the nearest local extremum close to the starting spproximation. The local optimization algorithms are further categorized

as gradient-independent such as downhill simplex or Powell and gradient-based algorithms such as least-squares fitting, steepest descent and Newton-Raphson methods. The global optimization methods can be classified as detereministic, stochastic and heuristic. The most successful deterministic optimization algorithms are inner approximation, outer approximation [5] and branch and bound [6, 7]. Stochastic optimization algorithms incorporate some probabilistic element in the objective function or the algorithm implementation. Some popular stochastic optimization algorithms are simulated annealing, stochastic tunneling and parallel tempering. The most popular global optimization algorithms are heuristic and metaheuristic algorithms, which includes evolutionary algorithms (genetic algorithms and evolutionary strategies), swarm-based optimization algorithms (particle swarm optimization and ant colony optimization) and tabu search.

However, all these methods have their intrinsic drawbacks. For example, the linear least-squares fitting method has many disadvantages when applied to parameter estimation for molecular dynamics simulations. These include poor extrapolation properties, limitation in the shapes that linear models can assume over long ranges, and sensitivity to outliers. Gradient-based methods are useful if the molecular properties in question are direct functions of the potential parameters. Unfortunately, in molecular simulations, the dependence of the cost function on the individual parameter values typically does not admit to analytical study, which prevents the use of gradient-based optimization methods. The genetic algorithms (GAs), an example of heuristic, stochastic optimization methods, are sometimes used by computational chemists to solve minimization problems such as conformational search, and molecular docking. The drawback of GA is that it can only be used for a parameter space with few dimensions and the best solution may not be chemically reasonable.

One class of algorithms to solve the optimization problems is direct search

algorithms. Direct search methods represent unconstrained optimization techniques that do not use derivatives. These algorithms include Nelder-Mead simplex algorithm, the multidirectional search methods and the Hook-Jeeves algorithm. The study of optimization methods that do not require the knowledge of gradients is an active research research area. The downhill (Nelder-Mead) simplex algorithm [8], a powerful and robust optimization method, is widely used for parameterization and has the advantage of being gradient-free. It belongs to a class of direct search methods [9, 10], i.e., methods that do not use derivatives. Simplex is advantageous as it requires evaluation of function at $d + 1$ points in a d dimensional parameter space, while other direct search methods evaluate function at more than $d+1$ points. It also has the advantage of being easily parallelized at levels distinct from cost function evaluation [11] . The parallel simplex has been implemented by various scientists in different ways for various applications [12, 13, 14]. The Nelder-Mead simplex method is a popular direct search method and it has been included as a standard featues in many commercially available software libraries incuding NAG, IMSL, and Matlab [15]. This method was successfully implemented by Faller et al. [16] for parameterization of molecular simulation force fields. Norrby and Liljefors [17] have also successfully used simplex with Newton-Raphson optimization to develop force-field parameters. Gaiddon et al. performed mono-objective and multiobjective optimization using simplex [18].

In many simulations, the observed outcome of the simulation includes a contribution from noise due to sampling error; a typical example would be molecular dynamics simulations of a thermodynamically averaged property. Because the sampling errors are non-systematic and independent, the variance of this noise in any averaged property decreases over time, so that the measurement gets more reliable with continued sampling. Thus the observed value of the objective function can be viewed as a deterministic, underlying value (that which would result from infinite

sampling), plus some incremental noise whose variance decreases with time. Thus, although the objective function can be very noisy and non-continuous, the underlying (noise-free) surface is relatively smooth.

The optimization technique used to find a combination of input parameters that generate the optimal value for an objective function that has random noise in the measurement is known as stochastic optimization [19]. There are various methods that can be used to solve the problem of optimization with noise in the function evaluation. One such approach is response surface methodology [20]. Box and Wilson [21] used this strategy to minimize a quadratic objective function perturbed by random noise of constant strength. Many researchers derived new optimization startegies using response surface methodology as a foundation. Barton and Ivey [22] implemented variation of the Nelder-Mead algorithm to deal with noisy function evaluations. Fan et al. implemented a stochastic response surface optimization via an enhanced Nelder-Mead simplex search procedure [23]. These authors tested Nelder-Mead variants on a suite of test problems using a stochastic noise term sampled from a truncated normal distribution which is added to the underlying function. Another famous approach to deal with noise in the objective function is stochastic approximation. This approach is used by Gilmore and Kelly [24] for an implicit filtering algorithm. Spall [25] [26] used a similar approach for stochastic approximaton algorithm development.

There are two approaches to understanding the behavior of optimization strategies: theoretical and empirical [27]. Theoretical study is concerned with obtatining proof of convergence under certain general conditions. This type of investigation is certainly useful, but often provides very little information for practitioners to choose one algorithm or the other. As demonstrated by Powell [28], there is hardly any correlation between the algorithms that enjoy guranteed convergence in theory and the algorithms that are actually used by practitioners. Empirical investigations eveluate

the performance of optimization algorithm on various standard objective functions, including discontinuous and multimodal functions. Factors like the initial state of the system and termination criterion can play an important role in the outcome of the optimization process.

The optimization process is computationally intensive and hence requires an automated procedure. During the last decade there has been considerable progress in the development of distributed computer systems using the power of multiple processors to efficiently solve complex, high-dimensional computational problems. The master-woker paradigm is the most popular environment used for parallel and distributed computing. In this design, a master process distributes the data sets to the workers for processing and workers return the result to the master. This design can be used with a wide variety of applications and is easy to manage.

1.1 Problem Statement

We consider the optimization of functions when each function evaluation is subject to a random noise. The precision of the function evaluation depends upon the time devoted to it, as additional computational efforts can be used to reduce the amount of noise through averaging. This may require thousands of CPU hours of simulations for some applications. An example is molecular modeling applications involving a model or force field. It is a common problem in computational chemistry to find the set of parameters that best describe the chemical and physical properties for a particular class of molecular system. The simulations that measure these properties are often sampling calculations, and thus are subject to a considerable degree of noise.

The objective function (that we also call the cost function) value at a point Λ_k in parameter space is thus assumed to be the sum of an underlying deterministic

function $f(\Lambda_k)$ and a random noise $\epsilon_k(t_k)$,

$$g(\Lambda_k) = f(\Lambda_k) + \epsilon_k(t_k), \tag{1.1}$$

where ϵ_k is distributed normally with mean zero and decreasing variance $\sigma_k^2(t_k) = (\sigma_k^0)^2/t_k$, such that

$$P(\epsilon, t) = \sqrt{\frac{t}{2\pi\sigma^2}} e^{\frac{-t\epsilon^2}{2\sigma_k^2}} \tag{1.2}$$

where t_k is the amount of time that the vertex Λ_k has been sampled. The inherent variance $(\sigma_k^0)^2$ may depend on the location in parameter space (some models may be noisier than others) but there is no expectation that this variance is known ahead of time.

The purpose of performing any simulation is mostly to calculate one of more physical properties of a system. Often, these properties correspond to experimentally observable physical properties of the system, such as pressure, density, temperature, diffusion coefficient, etc. The property p is calculated from the result of a simulation T, with results depending on the parameters $\Lambda_k : p[T(\Lambda_k)]$. In order to compare the calculated and experimantal properties, we define a cost function $g(\Lambda)$:

$$g(\Lambda) = \sum_{i=1}^{N_p} \frac{1}{w_i^2} \frac{[p_i(\Lambda) - p_i^0]}{(p_i^0)^2} \tag{1.3}$$

Where w_i represents the weight for the property p_i. Practitioners often choose fitting targets that are noiseless, easy and cheap computationally rather than choosing fitting targets which are more physically relevant but may be hard and expensive to investigate. Unfortunately, there is a random noise in the meaurement of cost function due to computer simulations and physical system measurement that alters the optimization process as the underlying algorithm gets the misleading information. There have

been very few efforts to study optimization algorithms in noisy environment[29].

Unlike classical deterministic problems where perfect information (and derivatives) is available about the cost function, there are many practical problems with no or little a priori information about the structure of the cost function as a function of parameter values.

The cost function is a highly non-linear function of parameter values and hence requires highly robust optimization techniques. A typical parameterization problem consists of regions of parameter space that deliver bad property values and highly sensitive regions where a slight change in parameter values results in large deviations in the cost function.

Furthermore, if the number of parameters to be optimized is large, the increased dimensionality of the optimization space requires a large amount of computational resources and computer time. Most practitioners make use of coarse-grained parallel structure by running multiple simulations as independent simulations on different processors. The parallel communication in this informal approach takes place via human intervention by manually evaluating the cost function for each simulation and restarting simulations again with different parameters.

Several difficult features of the optimization problem prevent the application of more sophisticated and automated optimization algorithms These include a cost function that is a well defined function of parameters, typically very expensive to evaluate, and whose cost can vary dramatically depending on the order of points evaluated.

1.2 Research Approach

We modifed the Nelder-Mead simplex algorithm for use in the optimization studies. The Nelder-Mead algorithm or simplex algorithm is well known algorithm for multidimensional unconstrained optimization. Though it is robust and does not need derivative information, still it is not accurate for optimizing the function containing noise. In a stochastic context, simplex can terminate inappropriately at a solution very far from the true optimum. Some researchers have developed new algorithms for stochastic optimization by modifying simplex [30].

We have developed stochastic variants of the simplex implementation to deal with the noise factor. We call these algorithms *maxnoise* and *point-to-point comparison* algorithm. As discussed in chapter 2, the maxnoise algorithm improves the precision of simplex in the late stages of optimization when the the vertices of simplex are close together both in parameter space as well as function value and the noise contribution becomes significant. The point-to-point algorithm improves the parameters by comparing the function values of only those vertices that have significant effect on the outcome of the simplex transformations. Each decision by this algorithm is made at a high statistical level of confidence by incorporating the expectation value of the noise in the comparison of function values. These new algorithms are compared with one existing stochastic algorithm and the original Nelder-Mead method in the computational study in chapter 3.

In our approach, we enhanced the master-worker (MW) framework developed at the University of Wisconsin [31] to handle communication between the vertices of simplex. MW is an object- oriented C++ library that helps users to develop master-worker type parallel applications easily for computational grids. The MW framework has already been used in the MetaNEOS project [32] to implement many grid based

parallel optimization solvers[33] [34] [35]. We added one more level to MW hierarchy by enabling each worker further initiate one MPI job leading to client-server type environment. All the complexity associated with performing heterogeneous simulations is handeled at the client-server level, leaving the job of decision making for the simplex with the master at the top level. Full details of the parallel implementation are provided in chapter 3. Using MPI for communication among the nodes provides a convenient, high-level abstraction at both the MW and client-server levels. The master collates the cost function $g(\Lambda)$ computed by the workers corresponding to each simplex vertex and decides the next transformation of the simplex. However, human input is still required to the extent of providing the initial set of parameters that initialize the vertices of the simplex before the subsequent optimization. The total cost of the optimization can depend dramatically on the initial state of the simplex, so it is not advisable to automate this step [22].

The portable programming framework MW was enhanced for parallel implementation of modified simplex to increase the efficiency. It has been shown that many scientific applications can be parallelized quiet efficiently for a grid setup by using the master worker paradigm[36] [37]. Moreover, Low communication between master and workers results in insignificant overhead. The framework and modifed simplex algorithms were tested by optimizing the Rosenbrock function. The results of these test are described in Chapter 3. Finally, the framework was used for TIP4P water model parameterization.

1.3 Optimization Methods

Optimization refers to the process of finding the best solution from a set of possible alternatives. The goal of optimization is to solve problems that seek to minimize or maximize an objective function by systematically choosing the values of input variables from within an allowed set. The objective function to be optimized may be a single objective function or a multi-objective function. In a single objective function optimization, an optimum is either a minimum or maximum, depending on our requirements. We generally define optimization problems as minimization problems and if the function is subject to maximizations,we simply minimize the negative of the function. Real world problems are not limited to finding the minimum or mamimum of single objective function but they are applied to a set of objective functions, each representing one criterion to be satisfied. Algorithms designed for such optimization problems are called multi-objective. One of the primary applications of optimization studies is to devise a theoretical model that accurately describes the behaviour of physical systems.Optimization then allows us to fit the parameters of such a model so that it is in a close agreement with the experimental results.

The optimization algorithms can be categorized on the basis of various features as described below:

1.3.1 Method of operation

Optimization algorithms are generally classified as deterministic or probabilistic algorithms. Deterministic algorithms do not use random number to decide the next step from the current state. There exists at most one way to proceed, otherwise the algorithm terminates. Deterministic algorithms always generate the same output

when the input is the same. Probablistic algorithms or randomized algorithms contains at least one instruction that acts on random numbers. With these algorithms, the specific output depends on the instantiation of a random process, as well as the inputs.

1.3.2 Properties

We can also distinguish optimization methods on the basis of time contraints, i.e. the required speed of the optimization algorithm. *Online optimization* refers to problems that require optimization in a time span between few millisecond to a few minutes. The optimum solution in this optimization may be sacrificed for speed gains. Robot localization and load balancing are examples of online optimization. The online optimization tasks are performed repetitively. *Offline optimization* does not impose strict time constraints on the task completion and hence optimization process may take even days to deliver a optimial or near-optimal solution.

1.3.3 Heuristic Methods

Heuristic algorithms represent a class of optimization algorithms that use the information currently gathered by the algorithm to decide which solution candidate should be tested next or how next candidate can be generated. Some popular heuristic algorithms are discussed below:

1.3.3.1 Genetic Algorithms

Genetic algorithms (GA) are a subclass of evolutionary algorithms that mimic the process of nature evolution. In a genetic algorithm, a population of strings also

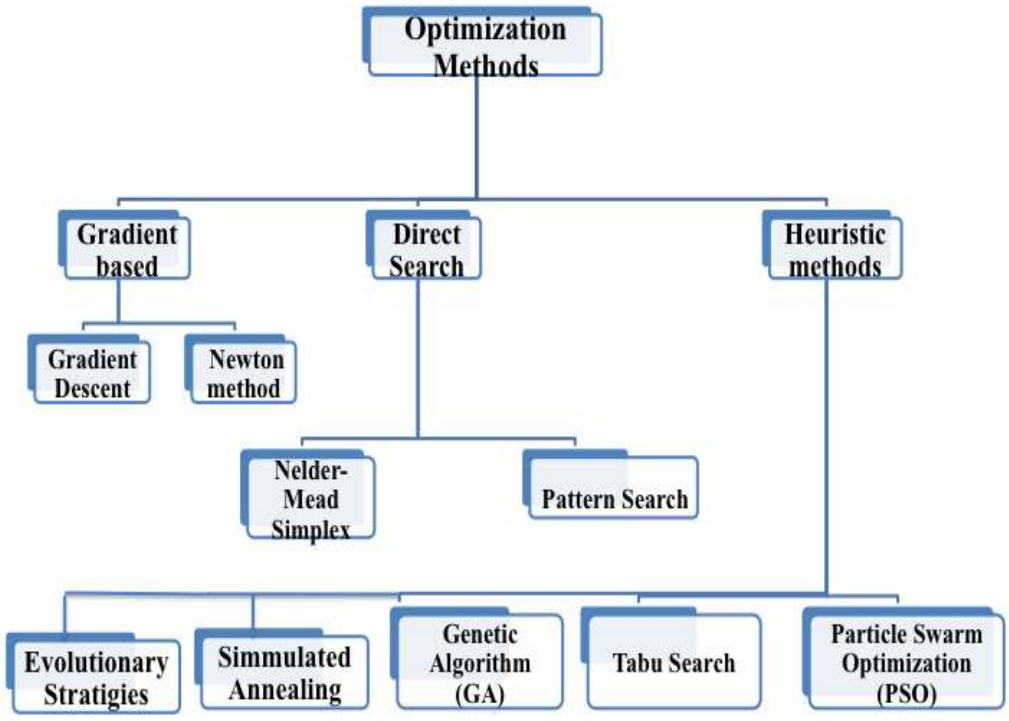


Figure 1.1: Optimization methods

known as chromosomes or a genotype encodes candidate solutions (phenotypes) to an optimization problem, and evolves towards the optimum solution. In other words, the value of the objective function is computed on the basis of phenotypes in the problem space generated by genotype-phenotype mapping. The solutions are generally represented in binary strings of 0s and 1s. The fitness of a member of the population is detemined by the objective function value. The members of population are subjected to selection, reproduction, crossover, and mutation [38]. It requires evaluation of several generations before a significant improvement in the objective function is seen.

Genetic algorithms are popular for their robustness in searching complex sur-

faces and find application in image processing, neworking and communication, economics and finance, combinatorial optimization, etc.

1.3.3.2 Evolutionary Strategies

Like GA, evolutionary strategies (ES) are based on the principles of natural evolution to solve parameter optimization problems. The first ES algorithm used a simple mutation-selection scheme called two-membered ES or (1+1)-ES [39]. The algorithm consisted of one parent producing one offspring by adding standard normal random variates. The better of parent and offspring becomes the parent for next generation. The algorithm stops when the termination criteria is met which includes the number of generations, elapsed CPU time, and absolute or relative progress per generation. The multinumbered ES, also known as $(\mu + 1)$-ES, has two parents randomly selected from a populaton of $\mu > 1$ parents, producing one offspring. Researchers have developed a hybrid GA-ES algorithm to solve multimodal continuous optimization problem [40].

1.3.3.3 Particle Swarm Optimization

Particle swarm optimzation (PSO) is a population-based stochastic optimization technique which simulates the behavior of biological social systems like a flock of birds or a school of fish. PSO shares many similarities with genetic algorithms (GA). In a PSO algorithm implementation, each particle keeps track of its coordinates in the problem space corresponding to the best solution it has obtained so far. It also keeps track of the best value obtained so far by any particle in the neighbourhood of this particle. When the particle considers the entire population as its neighbour then the best value is global best. PSO is an efficient global optimizer for continuous variable problems and can easily be implemented with few parameters to fine-tune. It can

easily be parallelized for concurrent execution. Though PSO has been successfully applied to various optimization problems like neural networks, structural optimization, and shape topology optimization, it suffers from the disadvantage of slow convergence in refined search stages, i.e. it has weak local search ability [41, 42, 43, 44, 45].

1.3.3.4 Simulated Annealing

Simulated annealing (SA) [46] is a global optimization method that can be applied to arbitrary search and problem spaces. Each step in SA replaces the current solution by an arbitary nearby solution, selected with probability that dependes upon the difference between the corresponding function values and also on a global parameter T, the temperature. The current solution changes randomly when T is large and moves downhill as T approaches zero. The ability of SA to move *uphill* saves the method from being stuck in local minima. One of the main reasons for the popularity of simmulated annealing algorithms, particularly for single objective optimization, is the existence of convergence proofs for this method. SA performs well on combinatorial problems and has several versions [47].

1.3.3.5 Tabu Search

Tabu search (TS) [48, 49] is a local search optimization method. TS algorithm marks all the coordinates that have already been visited as *tabu*. These solutions are not visted again thus reducing the chances of getting stuck in a local minima. This approach can be implemented using a list, which stores all the candidate solutions that have already been visited. If the newly created phenotype is found in this list, it is rejected. The list has maximum length n and any $n + 1$ solution candidate is added by removing the first element in the list. The simple Tabu Search is very similar to simulated annealing. Some example in which tabu search has been applied

are combinatorial optimization, machine learning, biochemistry, operation research, networking and communication.

1.3.3.6 Ant Colony

A recently proposed metaheuristic optimization approach called Ant Colony optimization (ACO), is used for hard combinatorial optimization problems [50, 51]. Ant colony optimization(ACO) is a probabilistic technique for problems that can be reduced to finding optimal paths through graphs [52]. ACO is based on the metaphor of ants that wander randomly in search of food and lay down a trail of pheromones. Once the ant has found some food, it can track its way back and also reinforce it with another layer of pheromones. Each ant that finds this path will follow it with certain probability and also excrete some pheromenes. The more longer the path, the more time pheromones have to evaporate. A short path gets traversed faster and thus pheromone density remains high. Many researchers have successfully implemented ACO using coarse-grained parallelization schemes [53, 54].

1.3.3.7 Response surface methodolgy

Response surface methodology (RSM) is a well known approach for modeling and analyzing engineering problems using statistical and mathematical models [55]. It is based on fitting a series of regression models to the output parameters of a simulation model and optimizing the regression model. The process is initiated with a first order regression and steepest ascent or descent search method and in the neighbourhood of the optimum, higher degree regression models are used. RSM is based on basic concepts of choosing an approximate model and the plan of experiments where the response has to be evaluated. RSM requires fewer number of simulation experiments as compared to many gradient-based algorithms.

1.3.4 Stochastic optimization

Stochastic optimization refers to finding the deterministic input parameters that generate the optimal value for a function that has random noise in the measurement [19]. Noise may be caused in the optimization problem because of measurement limitations, limited accuracy, or sampling errors in the simulation process. Stochastic optimization algorithms find a wide range of application in science, engineering, transportation, statistics and business [56]. Specific applications include industrial processes (making investment decisions in order to increase profit), aerospace engineering (running simulations to refine the design of a projectile or aircraft), medicine (running simulations to extract the maximum information about the efficacy of a new drug), and traffic engineering (controlling the timing for the signals in a traffic network).

The effects of noise in the optimization has been studied by many scientists including Miller and Goldberg [57] [58] , Lee and Wang [59], Sigrn Andradttir [60].

1.3.5 Direct search methods

The Direct search methods involve the comparision of each trial solution with the best available solution at that time. They do not require a numerical function value; the relative rank of function values is sufficient [61]. In other words, the direct approach needs less measurement data as the data becomes unnecessary. The direct search methods are also known as "derivative-free" as these methods never compute or approximate derivatives. These methods are recommended in a situation where the measurement of data is expensive or time consuming and data is contaminated by significant noise [27]. The direct search methods are easily parallelizable and have advantages of achieving global convergence in some problems where standard

Newton-like methods fail [62]. The Pattern search algorithm [63] and the downhill Nelder-Mead algorithm represent two different examples of direct search methods.

1.3.5.1 Simplex method

The downhill simplex algorithm for finding a local minimum of a function of several parameters was devised by Nelder and Meade [8]. The simplex itself is defined by $d+1$ vertices $\Lambda_1, \ldots, \Lambda_{d+1}$, i.e. one more than the d dimensions of the parameter space (a triangle in two dimensions, a tetrahedron in three dimensions, etc).

The simplex is moved iteratively through the parameter space based on the values of the objective function at each vertex. This is done by discarding one vertex and adding a new one using simple geometrical transformations: reflection, extension, contraction and collapse.

The simplex algorithm is widely used in many fields, especially in chemistry and chemical engineering [64]. It does have some inherent disadvantages that includes premature termination at the local minimum, high sensitivity to initial points and inability to cope with noisy systems. Though the original simplex method was designed to find the local minima of a real-valued convex function, it has also been used for finding the global minima of non-convex functions. This is done either by restarting the simplex or by using it as a local search subroutine with in a metaheuristic method [65, 66, 67, 68, 69, 70].

1.4 Document Organization

The document is organized as follows: Chapter 2 explains the simplex method and the proposed algorithms. Chapter 3 elaborates performance measurement of the proposed algortithms with other existing algorithms and also with each other. It also

explains the scale-up studies for optimizing the Rosenbrock function using simplex with the MW framework. We also discuss the application of simplex embedded in enhanced MW framework to TIP4P water model parameterization in this Chapter. Chapter 4 provides an overview of the work we have completed, software implementation of enhanced MW, and directory structure required to perform the simulations. Chapter 5 summarizes the work we have done and further scope of this research.

Chapter 2

Algorithms

2.1 Simplex Algorithm

The downhill simplex algorithm (Algorithm 1) due to Nelder Mead in d dimensions consists of $d+1$ vertices. A d-dimensional simplex is denoted with vertices $\Lambda_i, i = 1, ..., d+1$. The value of the objective function at Λ_i is represented by $g(\Lambda_i)$. Let Λ_{max}, Λ_{smax}, Λ_{min} represent the vertices with highest, second highest, lowest objective function value respectively. Λ_{cent} denotes the centroid of all the vertices except Λ_{max}. The simplex algorithm uses several transformation operations including reflection, expansion, and contraction to determine a new vertex to replace Λ_{max}. The algorithm replaces the worst vertex Λ_{max} with a new vertex generating a smaller function value after one of the possible operations. If none of these operations generate a vertex better than Λ_{max} then simplex will collapse towards Λ_{min}. The simplex operations are defined as follows:

- Reflection: $\Lambda_{ref} = (1+\alpha)\Lambda_{cent} - \alpha\Lambda_{max}$ with α the reflection coefficient ($\alpha = 1$).
- Expansion: $\Lambda_{exp} = \gamma\Lambda_{ref} - (1-\gamma)\Lambda_{cent}$ with γ the expansion coefficient ($\gamma > 1$).

- Contraction: $\Lambda_{con} = \beta\Lambda_{max} + (1-\beta)\Lambda_{cent}$ with β the contraction coefficient $(0 < \beta < 1)$.

For optimal performance of simplex α, β, γ are typically set to 1, 0.5, 2 respectively.

input : Initialize simplex Λ_i, $1 \leq i \leq d+1$
output: Minimum function value g(Λ_{min})
while *termination condition not fullfilled* **do**
  get Λ_{max}, Λ_{min}, Λ_{smax} , Λ_{cent}
  $\Lambda_{ref} = 2\ \Lambda_{cent}$ - Λ_{max} ; /* Reflection */
  if $g(\Lambda_{ref}) < g(\Lambda_{min})$ **then**
    $\Lambda_{exp} = 2\Lambda_{ref}$ - Λ_{cent} ; /* Expansion */
    if $g(\Lambda_{exp}) < g(\Lambda_{ref})$ **then**
      $\Lambda_{max} = \Lambda_{exp}$
    else
      $\Lambda_{max} = \Lambda_{ref}$
    end
  else
    if $g(\Lambda_{ref}) < g(\Lambda_{max})$ **then**
      $\Lambda_{max} = \Lambda_{ref}$
    else
      $\Lambda_{con} = 0.5\Lambda_{max} + 0.5\ \Lambda_{cent}$; /* Contraction */
      if $g(\Lambda_{con}) < g(\Lambda_{max})$ **then**
        $\Lambda_{max} = \Lambda_{con}$
      else
        do $i = 1, d+1$
        if $i \neq min$ **then**
          $\Lambda_i = 0.5\Lambda_i + 0.5\Lambda_{min}$; /* Collapse */
        end
      end
    end
  end
end

Algorithm 1: Deterministic simplex (DET)

Complexity of simplex

Since the ordering of vertices is performed in each step,the complexity of the simplex algorithm is $O(d \log d)$. Let $c(d)$ represents the number of operations required to calculate the objective function value $g(\Lambda)$, then the reflection, contraction, expansion expressions can be calculated in $O(c(d))$. Thus the total number of operations are bounded either by the ordering step or by $c(d)$. If the shrinking operation is performed then d is the number of operations performed in each iteration and the complexity is $O(d.c(d))$

Evaluating the objective function value at a given point requires exactly $O(d)$ operations while the worst-case complexity for each NM iteration is greater than or equal to $O(d^2)$. So ordering $d+1$ points is always less time consuming than calculating the objective function. Researcher are less focused on reducing the complexity of sorting step, rather more emphasis is given to reducing the function evaluations required for finding the objective function.

2.2 Max noise algorithm

Our goal for the optimization procedure is to find the minimum of the objective function,

$$\Lambda^* = \min\{g(\Lambda)|\Lambda \in R^n\}. \tag{2.1}$$

As in Anderson et al.[71] we note that the size of the simplex, whether defined as a hypervolume or a "diameter"

$$D(\Lambda) = \max_{j,k=1,2,..d+1} |\Lambda_j - \Lambda_k|, \tag{2.2}$$

is always a multiple 2^{-l} of the size of the initial simplex. Contraction steps halve the size of the simplex, incrementing the "contraction level" l by one, while expansion steps double the size of the simplex, decrementing l by one. Reflection steps leave the size and l unchanged, while collapse operations increase l by d. Tracking the size of the simplex is relevant because when the simplex is large, the objective function values $\{g(\Lambda_k)\}$ are likely to be quite different, and the effect of the noise on the motion of the simplex is small. On the other hand, when the simplex size is small, the points are close together both in parameter space and in function value, and the noise has a greater effect on the ordering of the vertices. Noise ϵ_k is distributed normally (eq 1.2) with mean zero and decreasing variance $\sigma_k^2(t_k) = (\sigma_k^0)^2/t_k$, where t_k is the amount of time that the vertex Λ_k has been sampled. The simplex algorithm has tendency to make wrong decisions in the presence of noise. Suppose the current decision based on the noise free underlying surface $g(\Lambda)$, is to reject Λ_{max} and replace it with the reflection, Λ_{ref}. However, because of the noise $g(\Lambda_{ref}) > g(\Lambda_{max})$, and reflection is rejected.

We introduce an additional condition in the simplex algorithm (Algorithm 1) to develop a new algorithm named max noise (Algorithm 2) (MN). The condition

$$\max_{i=1,\ldots,d+1}(\sigma_i^2(t_i)) > k\overline{(g(\Lambda_i) - \overline{g})^2}, \tag{2.3}$$

is intended to postpone the decision of simplex transformation until the noise at each of the vertices is small compared to the internal variance of the vertices, which allows the simplex to avoid making wrong decision due to noise in the objective function . Here $\overline{g}$ represents the average of $g(\Lambda_i)$ over all the vertices and k is a constant. All the notations used in Algorithm 2 have the same meaning as in Algorithm 1.

input : Initialize simplex Λ_i, $1 \leq i \leq d+1$
output: Minimum function value g(Λ_{min})

while *termination condition not fullfilled* **do**
  get Λ_{max}, Λ_{min}, Λ_{smax}, Λ_{cent}
  $\Lambda_{ref} = 2\ \Lambda_{cent}$ - Λ_{max} ; /* Reflection */
  while $\max_{i=1,\ldots,d+1}(\sigma_i^2(t_i)) > k\overline{(g(\Lambda_i) - \overline{g})^2}$ **do**
    /*noisiest vertex has a variance too much larger than the internal variance of the vertices themselves */
    Wait
  end
  if $g(\Lambda_{ref}) < g(\Lambda_{min})$ **then**
    $\Lambda_{exp} = 2\Lambda_{ref}$ - Λ_{cent} ; /* Expansion */
    if $g(\Lambda_{exp}) < g(\Lambda_{ref})$ **then**
      $\Lambda_{max} = \Lambda_{exp}$
    else
      $\Lambda_{max} = \Lambda_{ref}$
    end
  else
    if $g(\Lambda_{ref}) < g(\Lambda_{max})$ **then**
      $\Lambda_{max} = \Lambda_{ref}$
    else
      $\Lambda_{con} = 0.5\Lambda_{max} + 0.5\ \Lambda_{cent}$; /* Contraction */
      if $g(\Lambda_{con}) < g(\Lambda_{max})$ **then**
        $\Lambda_{max} = \Lambda_{con}$
      else
        do $i = 1, d+1$
        if $i \neq min$ **then**
          $\Lambda_i = 0.5\Lambda_i + 0.5\Lambda_{min}$; /* Collapse */
        end
      end
    end
  end
end

Algorithm 2: Max noise (MN)

This approach has the benefit that simulations run only for a short amount of time in the early stages of the optimization, where accurate sampling is not needed in order to reject poor parameter values, decreasing computational cost. On the other hand, in the late stages of the optimization, simulations are allowed to run as long as needed in order to distinguish between closely clustered parameter values, so that accuracy is not limited.

This algorithm is compared to a similar approach used by Anderson et al.[71], in which it is required that the standard deviation of the noise at each vertex be less than a cutoff which becomes more stringent as the simplex gets smaller,

$$\sigma_i^2(t_i) < k_1 2^{-l(1+k_2)}, \forall i \tag{2.4}$$

where k_1 and k_2 are arbitrary constants that must be specified.

The Anderson algorithm differs from the standard NM simplex approach. It operates on a set of m points and this set of points is known as *structure*.The size of strcture S is defined as

$$D(S) = \max_{j,k=1,2,..m} |x_j - x_k|, \tag{2.5}$$

New structures are generated from a given structure S with reflection or expansion around a point x, operations that differs from the similarly named operations on a simplex:

$$REFLECT(S, x) = \{2x - x_i | x_i \in S\} \tag{2.6}$$

$$EXPAND(S, x) = \{2x_i - x_i | x_i \in S\} \tag{2.7}$$

The expansion operation doubles the size of a structure while the contraction opera-

tion which reduces the size of structure to half and is defines as:

$$CONTRACT(S, x) = \{0.5(x + x_i | x_i \in S\} \tag{2.8}$$

Note that the algorithm originally presented by Anderson et al. is a direct search optimization approach that differs from the Nelder-Mead simplex algorithm; here we evaluate their convergence criterion (eq. 2.4), but do not adopt the other features of their method.

2.3 Point-to-point comparison algorithm

One of the drawbacks associated with the proposed maxnoise algorithm is that if one vertex has large noise, the internal variance of the function values of vertices may still be small causing the simplex to wait for a very long time due to the additional condition in max noise algorithm (line 4). The vertex that has large noise may not be one of those (lowest, highest, second high) that actually affect the motion of the simplex. One solution is to compare the significant vertices, i.e. vertices that actually move the simplex, one by one rather than requiring convergence at all vertices. Thus we modified the implementation of the downhill simplex algorithm by introducing such a point-to-point comparison of function values among the vertices of simplex. As in algorithm 1 , the objective function value at a point is given by equation 1.1. Each simplex decision is made at a higher confidence level by including the expectation value (one standard deviation away from mean) σ of the noise contaminating the function value contaminated by noise. In other words, comparison between function values in the simplex algorithm is is made more strict by requiring not just that $g(\Lambda_i) < g(\Lambda_j)$, but that $g(\Lambda_i) + k\sigma_i < g(\Lambda_j) - k\sigma_j$, where σ_i is the (expectation value

of the) statndard deviation of the noise at the vertex σ_i. That is, we require not just that the new vertex be lower but also that its $k\sigma$ confidence interval not intersect that of the vertex against which it is tested. Sampling proceeds until the point where the simplex transformation can be made at the chosen accuracy.

The value of σ decreases as the square root of time ($\sqrt{t}$) and hence the convergence criterion can be satisfied with additional sampling. Algorithm 3 describes the full procedure, in which σ_{max}, σ_{smax}, σ_{min} represent the value of σ at vertices with the highest, second highest and lowest objective function values, respectively.

The comparision of simplex vertices in this inplementation is performed at seven different stages of the simplex transformation. The algorithm can be optimized by investigating the different combination of comparison conditions. A complex implementation of this algorithm has point-to-point comparision with certain probabilities at all seven different stages while the most basic implementation requires point-to-point comparison at only one stage. These variations are considered in section 3.3. The point to point comparison algorithms can be modified by requiring the comparison to be made at varying confidence levels by adjusting the interval widths, i.e. by choosing the different values of k. This is considered in section 3.3. A variant of point-to-point comparison algorithm is implemented by allowing simplex decision made at even higher confidence level by including the expectation value (two standard deviation away from mean) σ of the noise containing the function value contaminated by noise.

One possible disadvantage of this method is that in cases where two vertices are coincidentally nearly identical, long sampling time will be required to determine which is lower, even though the eventual result may not depend strongly on the outcome.

input : Initialize simplex Λ_i, $1 \leq i \leq d+1$
output: Minimum function value g(Λ_{min})

while *termination condition not fullfilled* **do**
  get Λ_{max}, Λ_{min}, Λ_{smax}, Λ_{cent}
  $\Lambda_{ref} = 2\ \Lambda_{cent}$ - Λ_{max}
  if $g(\Lambda_{ref}) + k * \sigma_{ref} < g(\ \Lambda_{smax}) - k * \sigma_{smax}$ **then** `/* condition 1 */`
    if $g(\Lambda_{ref}) - k * \sigma_{ref} > g(\ \Lambda_{min}) + k * \sigma_{min}$ **then** `/* condition 2 */`
      $\Lambda_{max} = \Lambda_{ref}$
    else
      $\Lambda_{exp} = 2\Lambda_{ref} - \Lambda_{cent}$
      if $g(\Lambda_{exp}) + k * \sigma_{exp} < g(\ \Lambda_{ref}) - k * \sigma_{ref}$ **then** `/* condition 3 */`
        $\Lambda_{max} = \Lambda_{exp}$
      else if $g(\Lambda_{exp}) - k * \sigma_{exp} \geq g(\ \Lambda_{ref}) + k * \sigma_{ref}$ **then** `/* condition 4 */`
        $\Lambda_{max} = \Lambda_{ref}$
      else
        resample vertices and repeat until condition 3 or 4 is satisfied
      end
    end
  else if $g(\Lambda_{ref}) + k * \sigma_{ref} \geq g(\ \Lambda_{smax}) - k * \sigma_{smax}$ **then** `/* condition 5 */`
    $\Lambda_{con} = 0.5\Lambda_{max} + 0.5\ \Lambda_{cent}$
    if $g(\Lambda_{con}) + k * \sigma_{con} < g(\ \Lambda_{max}) - k * \sigma_{max}$ **then** `/* condition 6 */`
      $\Lambda_{max} = \Lambda_{con}$
    else if $g(\Lambda_{con}) - k * \sigma_{con} \geq g(\ \Lambda_{max}) + k * \sigma_{max}$ **then** `/* condition 7 */`
      do $i = 1, d+1$
      if $i \neq min$ **then**
        $\Lambda_i = 0.5\Lambda_i + 0.5\Lambda_{min}$
      end
    else
      resample vertices and repeat until condition 6 or 7 is satisfied
    end
  else
    resample vertices and repeat until condition 1 or 5 is satisfied
  end
end

Algorithm 3: Point-to-point comparison algorithm(PC)

2.4 Point-to-point with maxnoise

We also tested an algorithm which combined the maxnoise and point-to-point comparison algorithms (PC+MN) as shown in Algorithm 4. Here, both the maxnoise condition (Eq. 2.3) as well as the individual point-to-point comparisons must be satisfied in order for a simplex move to proceed. This implementation imposes stricter conditions on the movement of the simplex, slowing down the convergence but hopefully improving the accuracy of the algorithm.

2.4.1 Termination Criterion

We used two termination criteria in determining whether the simplex has converged sufficiently to be stopped. In the first of these, the simplex is terminated when all function values are within a predefined tolerance,

$$\max_i |g\{\Lambda_i\} - g\{\Lambda_{min}\}| \leq \delta \tag{2.9}$$

In the second termination condition, the optimization was terminated if the total walltime exceeded a predetermined limit. If either termination condition was satisfied, the simplex was stopped.

input : Initialize simplex Λ_i, $1 \leq i \leq d+1$
output: Minimum function value $g(\Lambda_{min})$

while *termination condition not fullfilled* **do**
  get Λ_{max}, Λ_{min}, Λ_{smax}, Λ_{cent}
  while $\max_{i=1,\ldots,d+1}(\sigma_i^2(t_i)) > k\overline{(g(\Lambda_i) - \overline{g})^2}$ **do**
    /*noisiest vertex has a variance sufficiently larger than the internal variance of the vertices themselves */
    Wait
  end
  $\Lambda_{ref} = 2\ \Lambda_{cent}$ - Λ_{max}
  if $g(\Lambda_{ref}) + \sigma_{ref} < g(\ \Lambda_{smax}) - \sigma_{smax}$ **then** /* condition 1 */
    if $g(\Lambda_{ref}) - \sigma_{ref} > g(\ \Lambda_{min}) + \sigma_{min}$ **then** /* condition 2 */
      $\Lambda_{max} = \Lambda_{ref}$
    else
      $\Lambda_{exp} = 2\Lambda_{ref} - \Lambda_{cent}$
      if $g(\Lambda_{exp}) + \sigma_{exp} < g(\ \Lambda_{ref}) - \sigma_{ref}$ **then** /* condition 3 */
        $\Lambda_{max} = \Lambda_{exp}$
      else if $g(\Lambda_{exp}) - \sigma_{exp} \geq g(\ \Lambda_{ref}) + \sigma_{ref}$ **then** /* condition 4 */
        $\Lambda_{max} = \Lambda_{ref}$
      else
        resample vertices and repeat until condition 3 or 4 is satisfied
      end
    end
  else if $g(\Lambda_{ref}) + \sigma_{ref} \geq g(\ \Lambda_{smax}) - \sigma_{smax}$ **then** /* condition 5 */
    $\Lambda_{con} = 0.5\Lambda_{max} + 0.5\ \Lambda_{cent}$
    if $g(\Lambda_{con}) + \sigma_{con} < g(\ \Lambda_{max}) - \sigma_{max}$ **then** /* condition 6 */
      $\Lambda_{max} = \Lambda_{con}$
    else if $g(\Lambda_{con}) - \sigma_{con} \geq g(\ \Lambda_{max}) + \sigma_{max}$ **then** /* condition 7 */
      do $i = 1, d+1$
      if $i \neq min$ **then**
        $\Lambda_i = 0.5\Lambda_i + 0.5\Lambda_{min}$
      end
    else
      resample vertices and repeat until condition 6 or 7 is satisfied
    end
  else
    resample vertices and repeat until condition 1 or 5 is satisfied
  end
end

Algorithm 4: Point-to-point with Maxnoise algorithm (PC+MN)

Chapter 3

Work Completed

3.1 Work Completed

In our parallel implementation of the downhill (Nelder-Mead) simplex algorithm in d dimensions, objective function evaluations must be kept active on each of the $d + 1$ vertices until it is certain that they are no longer needed. In addition, prospective objective function evaluations are also needed at the reflection or contraction, and perhaps also an extension, before it is known which vertex will be discarded. Thus a maximum of $d + 3$ vertices may be active at any one time.

Computationally, the parallel communications are implemented using a master-worker (MW) architecture, in which the computation is broken up into a collection of independent tasks, which are assigned to individual worker processes by the master process. The master process is logically associated with the simplex object, and performs all of the decision making for the optimization, while each worker is logically associated with a vertex object, and the tasks correspond to the evaluation of an objective function value at each point in parameter space. Tasks and workers do not communicate with one another directly, but report results to, and receive instructions

from the master. The master has the ability to direct a cessation of work at one point in parameter space and the initiation of new simulations at a different point. We use a modified version of the MW code developed by the University of Wisconsin [31] to coordinate the communication. This is an object-oriented set of C++ libraries that

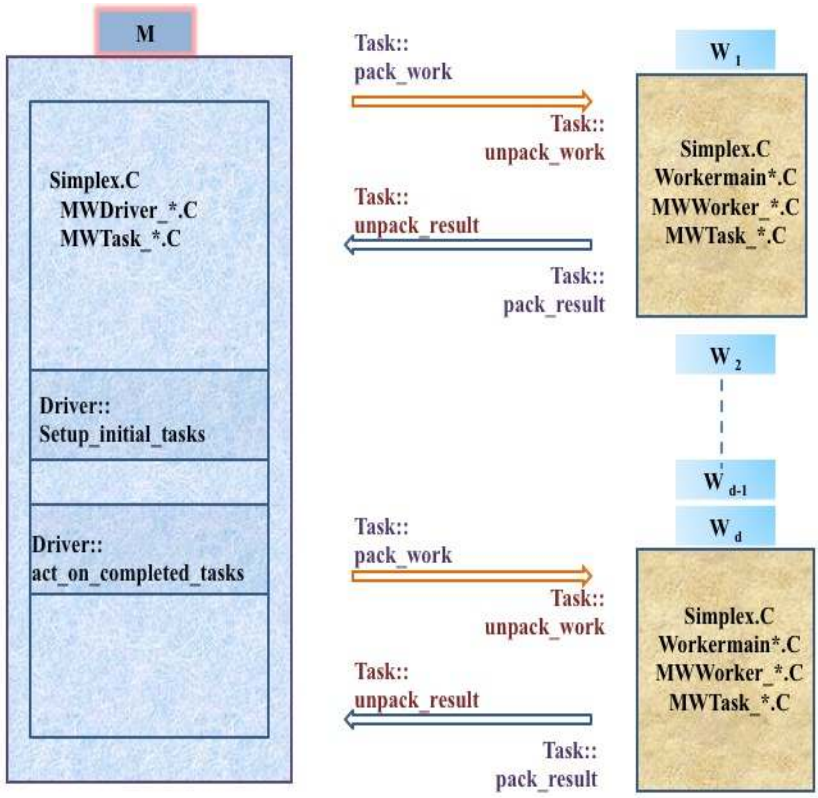


Figure 3.1: MW software implementation

provides abstraction to master, worker and task entities. We re-implemented three major classes of MW, namely MWDriver, MWWorker, and MWTask (Figure 3.1). The MWDriver manages a set of workers to execute the tasks. MWWorker class functions execute worker tasks, compute results, report results back, and wait for another task. MWTask stores the data describing the task and the results computed by the workers. It is the abstraction of one unit of work. These three abstract base classes hide the difficulties associated with the metacomputing and allow rapid development

of the scientific computing application. The communication functionality between the master and worker is implemented using abstract class MWRMComm virtual functions: *pack(⟨type⟩ array, int size)*, *unpack(⟨type⟩ array, int size)*, *send(int to-whom, int message-tag)*, and *recv(int from-whom, int message-tag)*. The MW program has the capability of using multiple different communication protocols, including sockets, file I/O, Condor/PVM and MPI. In our implementation, we use MPI communication between master and workers.

Each evaluation of an objective function is itself best treated as a parallel process. This is because a number of simulations may be needed in order to determine all of the properties needed to evaluate a given set of parameters. For example, separate simulations may be needed to evaluate the room-temperature energy, the isothermal compressibility, and the high-temperature properties of a given model, and these sampling simulations themselves can each be run in parallel. Consequently, each of the workers corresponding to one vertex of the simplex is logically identified with a second (server) process running in a completely different MPI environment. The workers and their corresponding servers communicate via file I/O (Figure 3.2).

Each of the N_s simulations associated with the vertex runs as its own (client) processes, in the same MPI environment as its server. These simulations can be efficiently implemented in parallel as there is no inherent correlation among them. We use the terminology of servers and clients at this lower level of implementation to distinguish these processes from those at higher level of simplex implementation. Communication between the server and its own clients occurs via MPI.

The parallel parameterization algorithm thus consists of 1 master communicating via MPI with $d + 3$ workers at the simplex level. At the vertex level, each of these $d + 3$ vertices initiates $N_s + 1$ processes forming an MPI job: 1 server and N_s clients. Thus in total there are $d + 4$ MPI jobs and $dN_s + 3N_s + 2d + 7$ processes

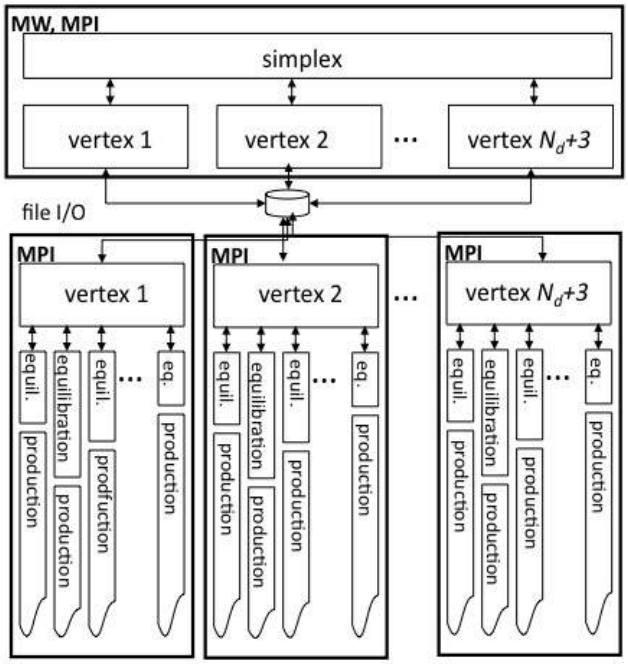


Figure 3.2: MW architecture.

(representing the 1 master, $d+3$ workers, $d+3$ servers, and $(d+3)N_s$ clients). The maximum number of cores consumed in this implementation is $dN_s + 3N_s + 2d + 7$. Since most of the CPU cycles are consumed by the simulation rather than the simplex logic as bookkeeping operations an efficient and advanced implementation could use as few as $(d+3)N_s$ cores without affecting the throughput significantly. This would require a considerably more complex code and is not an approach used in the current implementation of the parallel parameterization.

3.2 Performance Measurement of MN algorithm

The performance of MN algorithm was compared with the method of Anderson et al. [71]. In this phase of testing, we used the Rosenbrock "banana" function (Figure 3.3) in three dimensions [72] as a test case to evaluate the performance of the extensions of the simplex algorithm in the presence of noisy data. This function is a common choice for testing local optimization algorithms, because it discriminates well between different methods: there is a long, narrow, banana-shaped valley in which the minimum is located, and making progress along this valley can be difficult.

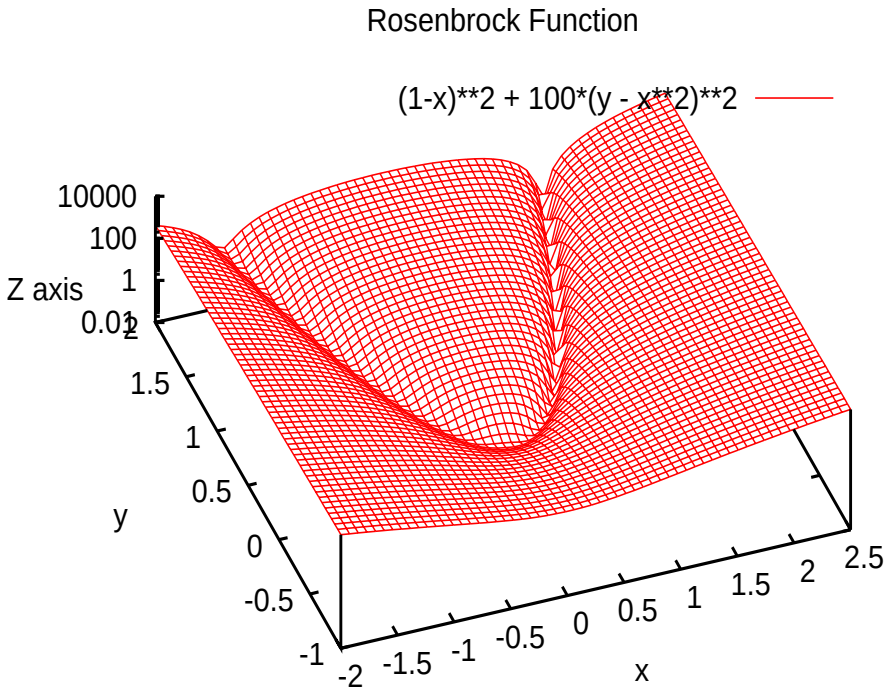


Figure 3.3: Rosenbrock function.

The function values for this test were directly computed from

$$f(\Lambda) = \sum_{i=1}^{2}(1 - \Lambda_{i-1})^2 + 100(\Lambda_i - \Lambda_{i-1}^2)^2, \tag{3.1}$$

which has a local minima at f(1,1,1)=0. Artificial Gaussian noise was added, with a variance inversely proportional to the duration for which the vertex had been active, as described in Eq. 1.2. In order to ensure that the optimization progress was limited by the level of noise, the parameter σ^0 was chosen so that simplex updates would occur on timescales of $\sim 10^4$ seconds in the late stages of the optimization. We studied both the algorithms with five different initial states generated by a random number generator, such that each of the three coordinates for each of the four vertices was uniformly distributed over $[6, -3]$.

Table 3.1: Results of optimization using MN algorithm with controlled noise.

input	N				R				D			
	k=2	k=3	k=4	k=5	k=2	k=3	k=4	k=5	k=2	k=3	k=4	k=5
1	76	323	183	103	1.15	1.68	1.82	1.22	1.43	1.63	1.65	1.47
2	126	142	326	169	8.08	7.90	7.97	7.88	5.19	5.05	5.09	5.00
3	115	569	175	99	2.23	2.9	2.88	2.50	1.79	1.90	1.92	1.86
4	95	187	441	70	.42	3.4	2.21	.15	1.02	1.92	1.79	.90
5	97	68	298	193	.02	.05	.079	.01	.51	.44	.49	.13

We optimized the algorithms themselves by evaluating four different values of the parameters k in eq 2.3 and k_1 in eq 2.4, while the value of Anderson's k_2 was always set to zero. Following Anderson, we used three separate performance measures to evaluate the success of the stochastic optimization procedures[71]. These are:

Table 3.2: Results of optimization using Anderson algorithm with controlled noise.

input	N				R				D			
	$k_1=2^0$	2^{10}	2^{20}	2^{30}	$k_1=2^0$	2^{10}	2^{20}	2^{30}	$k_1=2^0$	2^{10}	2^{20}	2^{30}
1	18	38	152	96	5.06	2.23	1.30	1.35	1.34	1.69	1.50	1.52
2	25	46	88	167	46.7	8.03	7.90	8.02	5.07	5.09	5.09	5.09
3	18	43	60	275	14.86	3.00	2.90	2.98	2.17	1.93	1.91	1.93
4	29	50	149	285	22.4	0.36	.12	.24	.32	.92	.58	.78
5	20	42	116	206	2.8	.14	.11	.126	1.22	.55	.60	.54

(1) The average number of iterations (N) required to reach convergence, (2) The mean error in the function value at convergence (R), and (3) The average distance of the lowest point in the simplex to the solution at convergence (D).

From the results shown in Table 3.1 and Table 3.2 we conclude that the accuracy of the optimization (given by R) under the MN algorithm is independent of the parameter k and is comparable to Anderson algorithm in the limit of large k_1. Also for Anderson algorithm we noticed that overly small values of parameter k_1 generate large errors (R). As shown in Figure 3.4 for the Anderson algorithm (RHS subfigures), for very small value of k_1, Anderson algorithm finds minimum values very far from the true minimum. This is attributed to the looser criterion for advancing the simplex, leading to too much contraction and premature convergence, as indicated by the smaller number of iterations, N. For large values of k_1, the Anderson algorithm is comparable to the MN algorithm (LHS subfigures). It is also obvious from these subfigures that for large values of k_1 both the algorithms take equal time to converge to a minimum. Since the the outcome of the objective function value is affected by value of k_1, Anderson algorithm requires that some additional computational efforts

be invested to tune the algorithm for every new problem. Thus method of Anderson *et al.* has the disadvantage that it must be parameterized separately for each new surface to be optimized, in order to find the factor needed to convert the simplex diameter to function noise such conversions are not needed in the MN algorithm. Also, for the MN algorithm, value of k does not affect the outcome of the algorithm. It only controls the speed of convergence. A small value of k in the range 1 to5 is appropriate. In case of Anderson algorithm, a small value of k_1 generates large error due to strict criterion while very large value of k_1 makes the size of simplex irrelevant and hence there is a possibility of high errors. The values of k_1 shall be proportional to the intial simplex size. A small k_1 for smaller simplex and large k_1 for larger simplex.

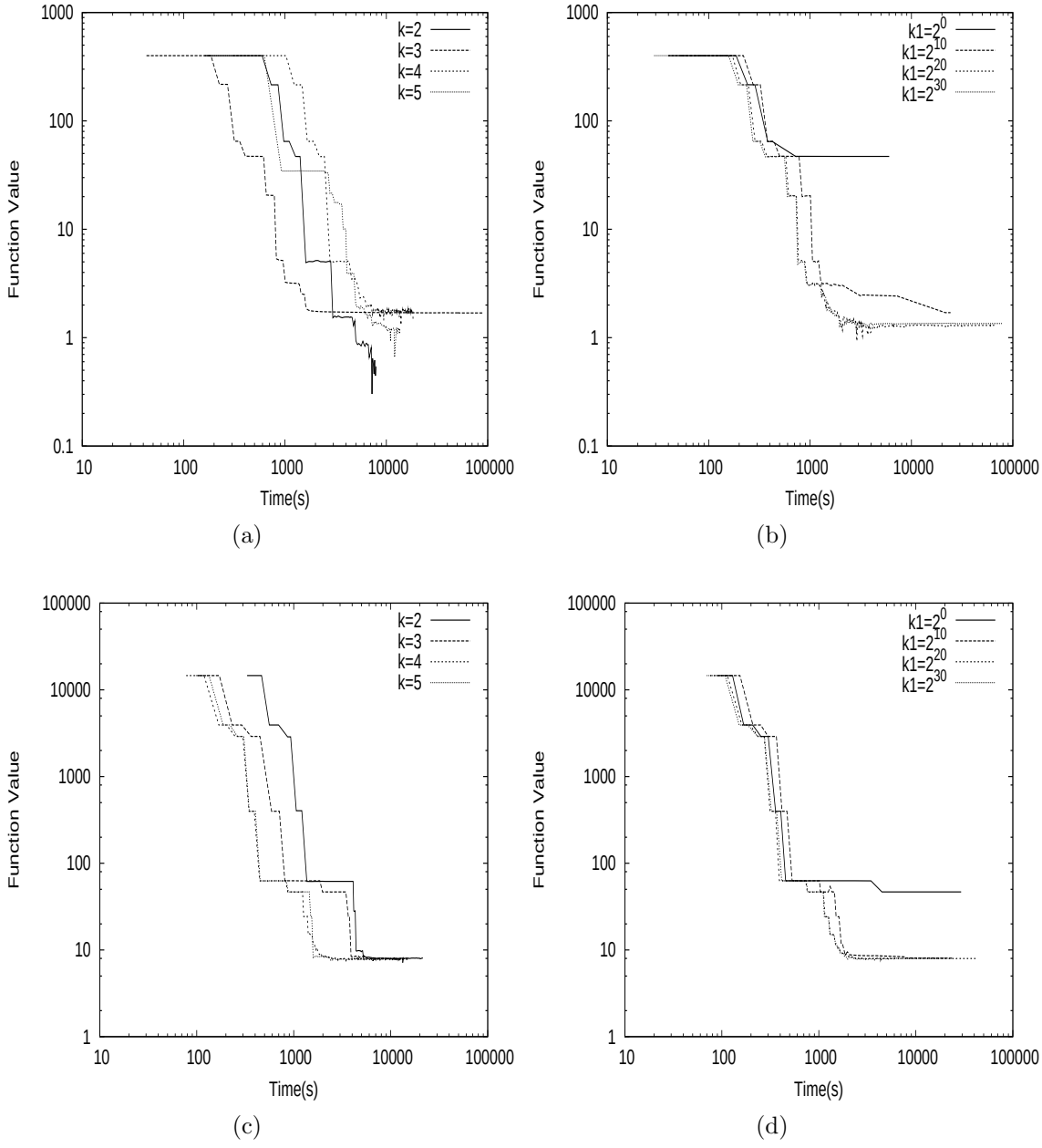


Figure 3.4

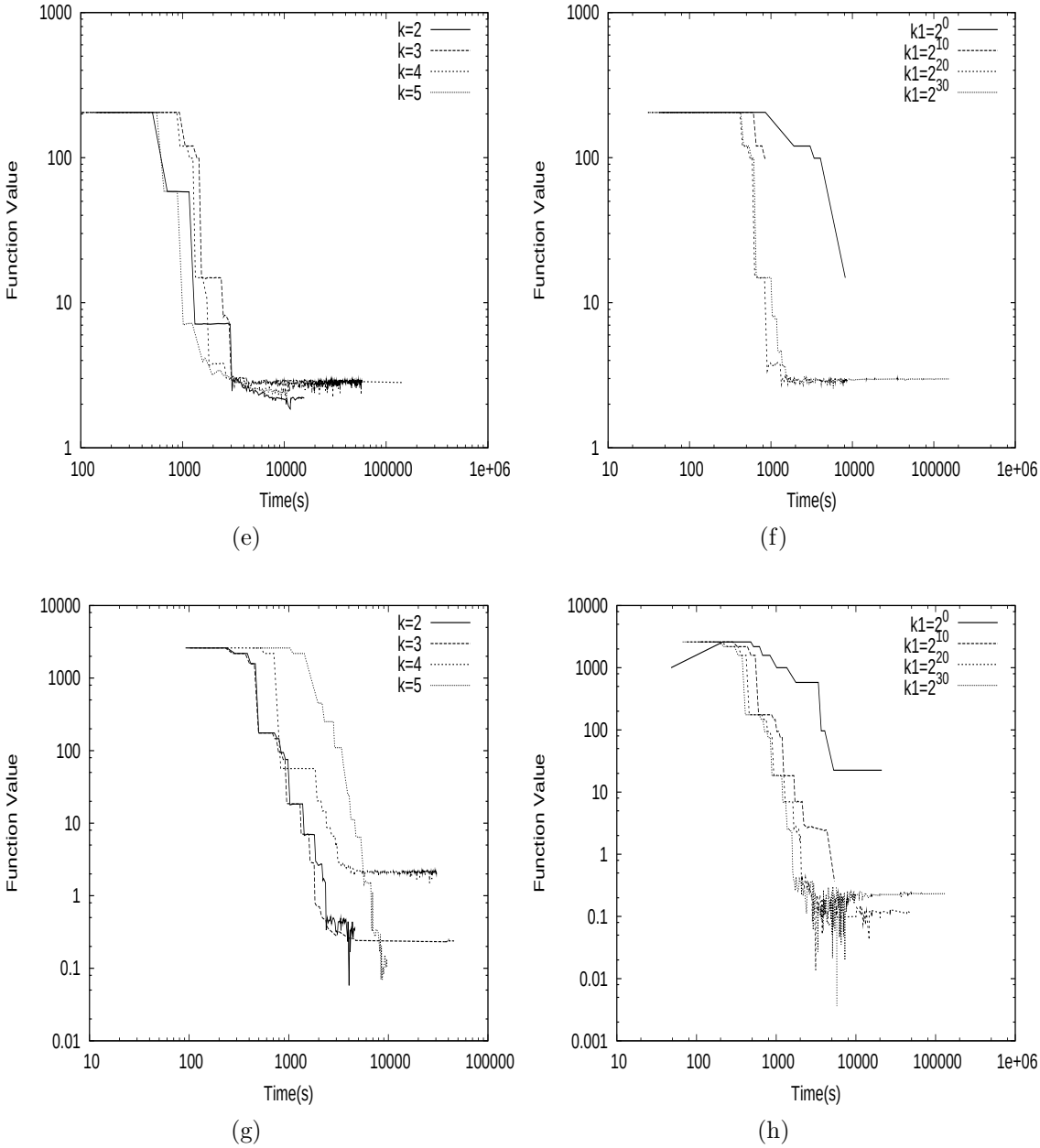
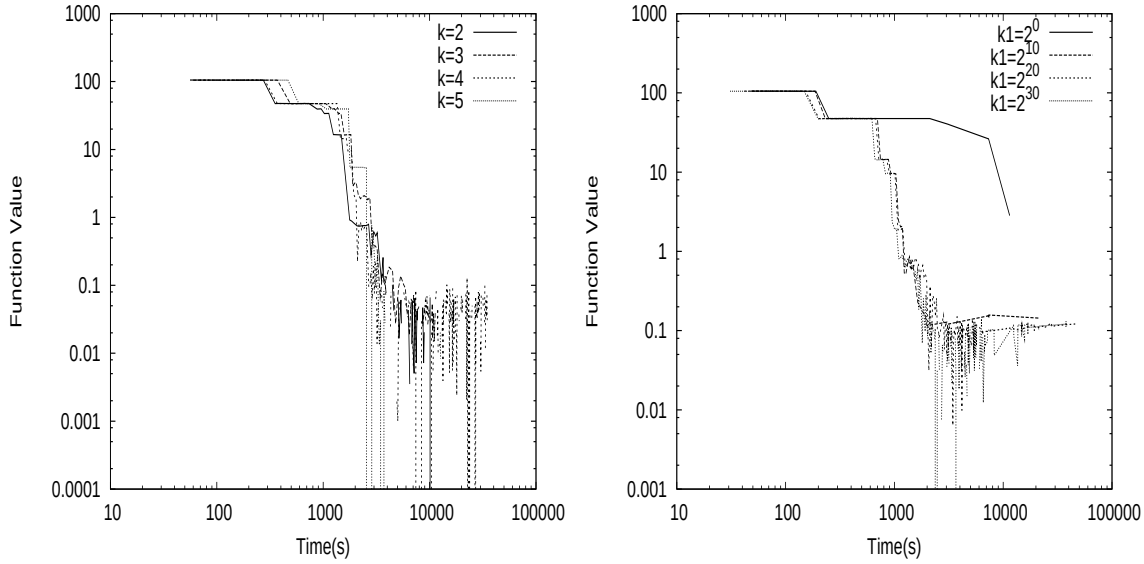

1000
100
10
1
Function Value
100
1000
10000
100000
1e+06
Time(s)
k=2
k=3
k=4
k=5
(e)
1000
100
10
1
Function Value
10
100
1000
10000
100000
1e+06
Time(s)
k1=2^0
k1=2^10
k1=2^20
k1=2^30
(f)
10000
1000
100
10
1
0.1
0.01
Function Value
10
100
1000
10000
100000
Time(s)
k=2
k=3
k=4
k=5
(g)
10000
1000
100
10
1
0.1
0.01
0.001
Function Value
10
100
1000
10000
100000
1e+06
Time(s)
k1=2^0
k1=2^10
k1=2^20
k1=2^30
(h)

Figure 3.4



(i) (j)

Figure 3.4: Function Value vs time for MN algorithm (left) and Anderson algorithm (right) with five different inputs. Each input tested with k=2,3,4,5 for MN algorithm and k_1=$2^0, 2^{10}, 2^{20}, 2^{30}$ for Anderson algorithm

3.3 Performance Measurement of PC and PC+MN

We also tested the DET, MN, PC, and PC+MN algorithms by optimizing the Rosenbrock function (eq 3.2) and the Powell function (eq 3.3) in 4 dimensions. The function values for this test were directly computed from the Rosenbrock function

$$f(\Lambda) = \sum_{i=2}^{4} (1 - \Lambda_{i-1})^2 + 100(\Lambda_i - \Lambda_{i-1}^2)^2, \tag{3.2}$$

which has a local minimum at f(1,1,1,1)=0 and the Powell function

$$f(\Lambda) = (\Lambda_1 + 10\Lambda_2)^2 + 5(\Lambda_3 - \Lambda_4)^2 + (\Lambda_2 - 2\Lambda_3)^4 + 10(\Lambda_1 - \Lambda_4)^4 \tag{3.3}$$

Artificial Gaussian noise was added to the Rosenbrock function and Powell function, with a variance inversely proportional to the duration for which the vertex had been active, as described by Eq. (1.2). In order to ensure that the optimization progress was limited by the level of noise, the parameter σ^0 was chosen so that simplex updates would occur on timescales of many thousands of seconds in the late stages of the optimization. Three different values ($\sigma^0 = 1$, 100, and 1000) were examined to study the effect of noise. For each noise level, each of the three algorithms was evaluated with 100 different initial simplex states generated by a random number generator, such that each of the four coordinates for each of the five vertices was uniformly distributed over $[5, -5)$.

The perfomance of these algorithms is compared in figure 3.5 and figure 3.6 for Rosenbrock and Powell function optimization respectively, which shows a distribution of the ratios of the minimum function value ($g(\Lambda)$) obtained by a pair of methods. These ratios are presented on a logarithmic scale, so a value of zero means that the two methods performed equally, and negative values mean that the method in the numerator of the ratio came closer to the minimum function value (of zero). For example, Fig. 3.6a shows that at low levels of noise, the maxnoise (MN) algorithm performs comparably to the standard deterministic simplex algorithm (DET) in the majority of cases, as the distribution is centered around zero. However, at higher noise levels the distribution acquires a progressively bigger tail at negative values, indicating that the MN algorithm avoids converging prematurely and attains a minimum function value that is lower by a factor of 10 to 10^4 in a significant minority of cases. By making some attempt to make simplex moves only when the noise level is small compared to the difference between vertices, the MN algorithm makes fewer incorrect moves and converges closer to the true minimum.

Likewise, Fig. 3.6b shows that the point-to-point comparison (PC) algorithm

ties or outperforms MN about 90% of the time. Particularly as the noise level increases, the PC algorithm is able to find minimum function values that are a factor of 10 or more better than those found by MN in the nearly half of the cases. This improvement results from the fact that the PC algorithm converges each vertex well enough to be confident in the specific simplex moves required, without overconverging unnecessarily.

The distribution in Fig. 3.5c is more symmetric, indicating that the PC+MN and PC methods are comparable. The distribution gets broader as the noise level increases, indicating that the behavior of the algorithms gets less predictable as the noise becomes stronger. The PC+MN algorithm performs slightly better at all noise levels, but only by a small margin. Although the end results are similar, the PC+MN algorithm is more effective in the sense that achieves this result with fewer simplex steps. This is evident from the fact that the PC+MN algorithm required 178 simplex steps on average at the high noise level, and 167 simplex steps at the intermediate noise level, compared to 900 (high) and 1082 (intermediate) for the PC algorithm, under the same termination criteria. By imposing stricter conditions on the motion of the simplex, the PC+MN algorithm spends more time sampling each vertex and takes fewer steps, but achieves a slightly more accurate minimum as a result.

The performance of the PC and PC+MN algorithms can be optimized by fine tuning which of the seven conditional comparisons in these algorithms use the stricter comparison criterion. This is done by letting the simplex make a decision by including the expectation value of noise (σ) for some of its transformation operations while ignoring it for all other operations.

Also both PC and PC+MN algorithms can be tested by forcing simplex to make a transformation with even higher confidence. This is done by choosing $k = 2$ in PC algorithm implementation. As a consequence of this implementation, we expect

simplex to make fewer incorrect transformations towards the optimum with higher confidence as it spends more time waiting for the noise to drop lower. This advantages may be mitigated to some extent because simplex may not make a move as the conditions become diffucult to satisfy and it may get stuck in one of the stages of PC algorithm.

We tested the variations of PC algorithms by choosing different combinations of comparisons to use the stricter (noise-aware) , using the Rosenbrock function. The optimization was performed with $\sigma^0 = 1000$. Results are shown in Figures 3.8 - 3.17. The following conclusions can be drawn from these tests:

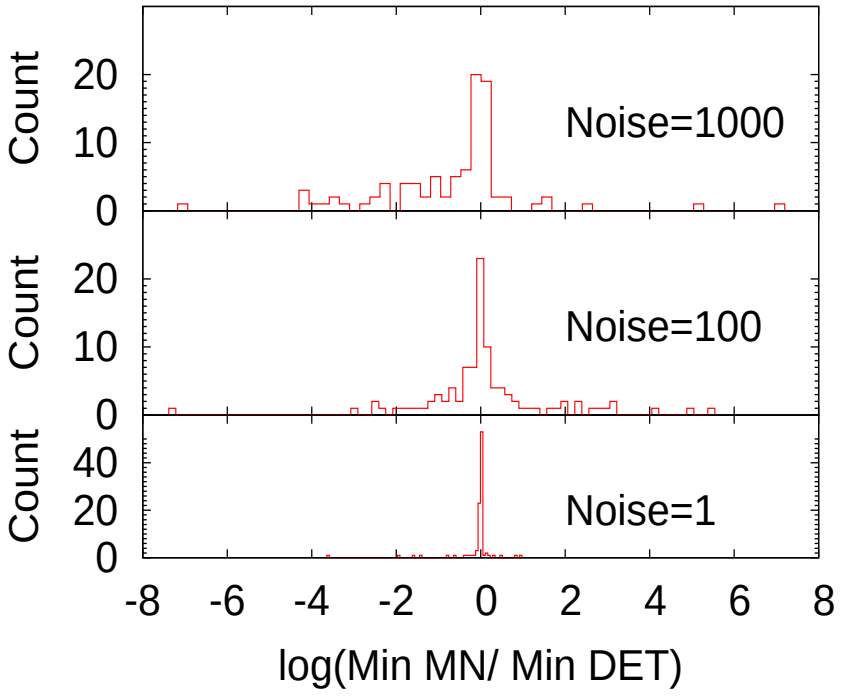
1) By increasing the confidence interval level ($k = 2$) in the PC algorithm implementation, no substantial change in the performance was observed for the Rosenbrock optimization as evident from Figure 3.7.

2) When the expectation value is considered only in condition 1 (c1) i.e. reflection step or condition 6 (c6) i.e. contraction step, one at a time, and compared with each other, the PC algorithm performs better in former as shown in Figure 3.8. This further leads to conclusion that when choosing only one condition, the choice of condition does make a difference on the simplex convergence i.e. all conditions are not equal.

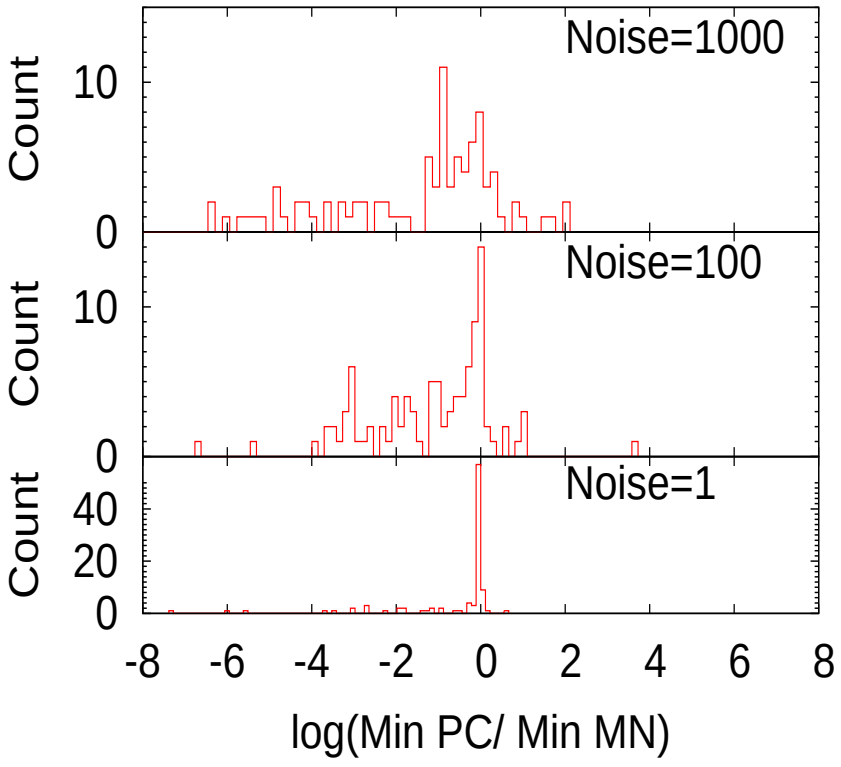
3) All conditions imposed together (c1-7) are too strict and include some harmful comparisons as is obvious from Figures 3.9 - 3.15. Any single condition is better than c1-7.

4) All are good, but c1, c2, and c3 are best, c4, c5, c6, and c7 are just moderate improvement over c1-7.

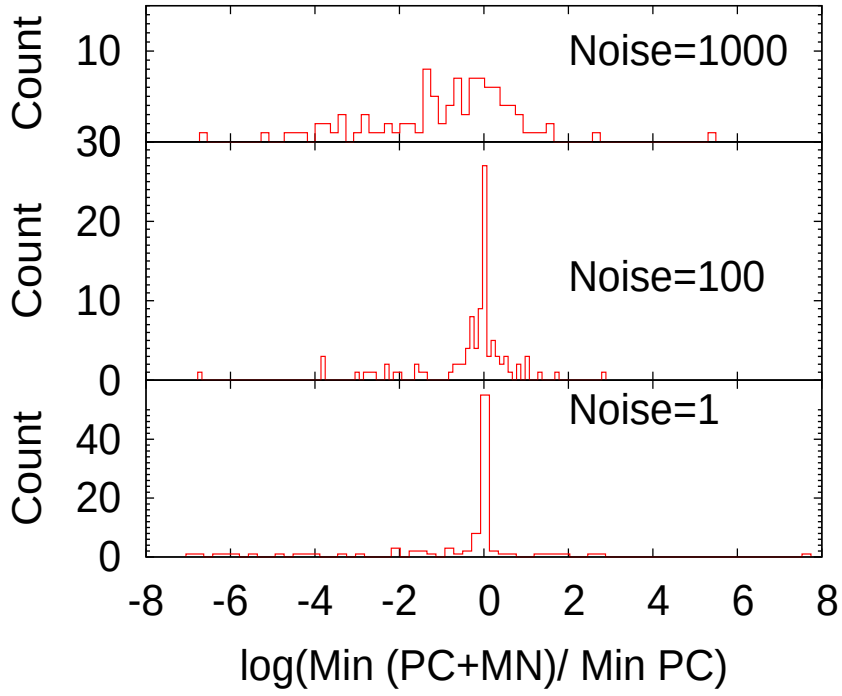
5) Combining a few key conditions, i.e. c136 (reflection, expansion and contraction) can also be better than c1-7 (Figure 3.17) but not as good as single condition only (Figure 3.16).


Count
20
10
0
Noise=1000
Count
20
10
0
Noise=100
Count
40
20
0
Noise=1
-8
-6
-4
-2
0
2
4
6
8
log(Min MN/ Min DET)

(a)

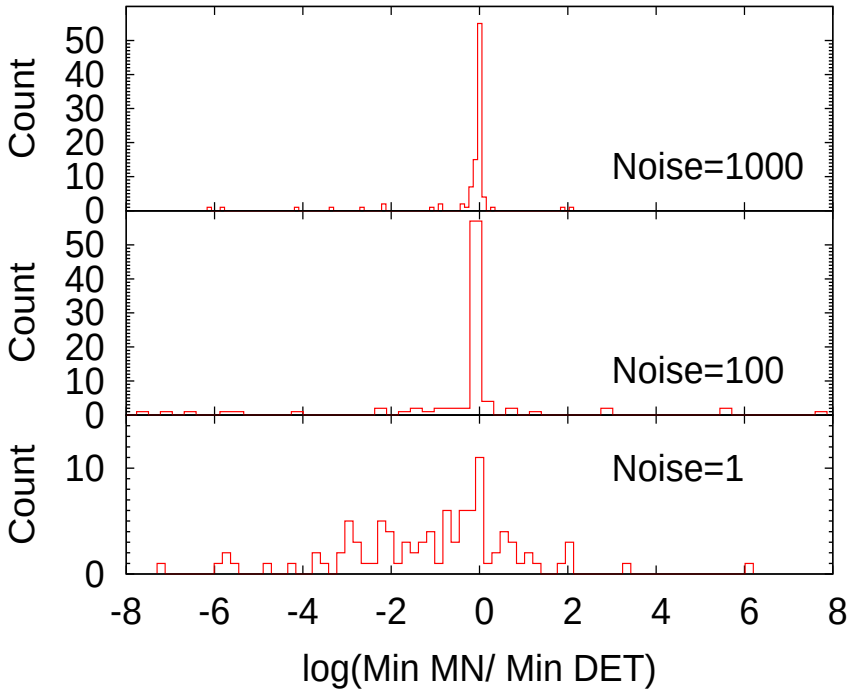

Count
10
0
Noise=1000
Count
10
0
Noise=100
Count
40
20
0
Noise=1
-8
-6
-4
-2
0
2
4
6
8
log(Min PC/ Min MN)

(b)

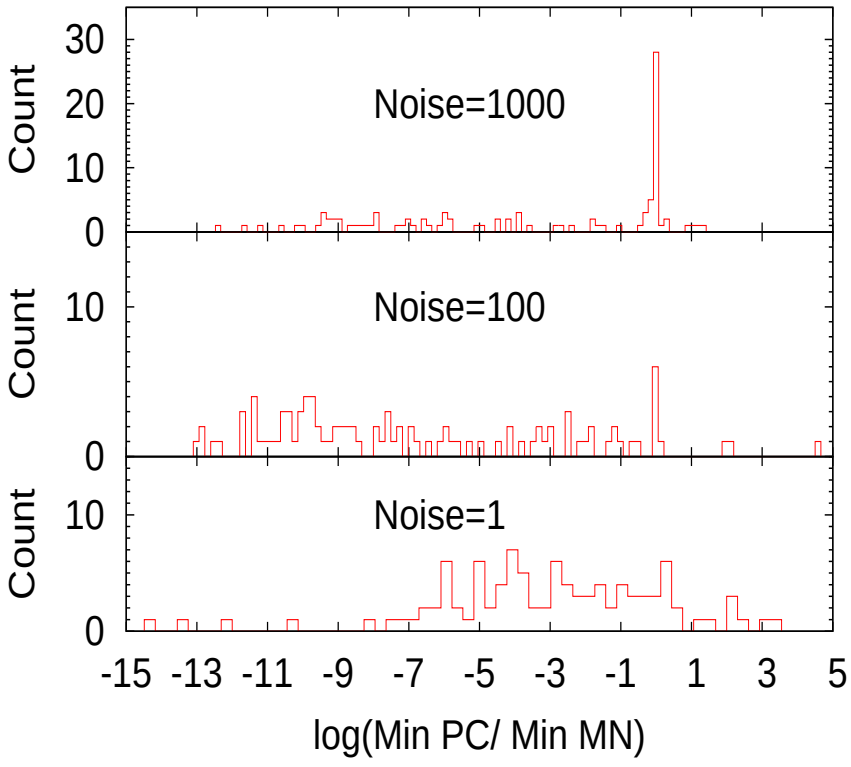


(c)

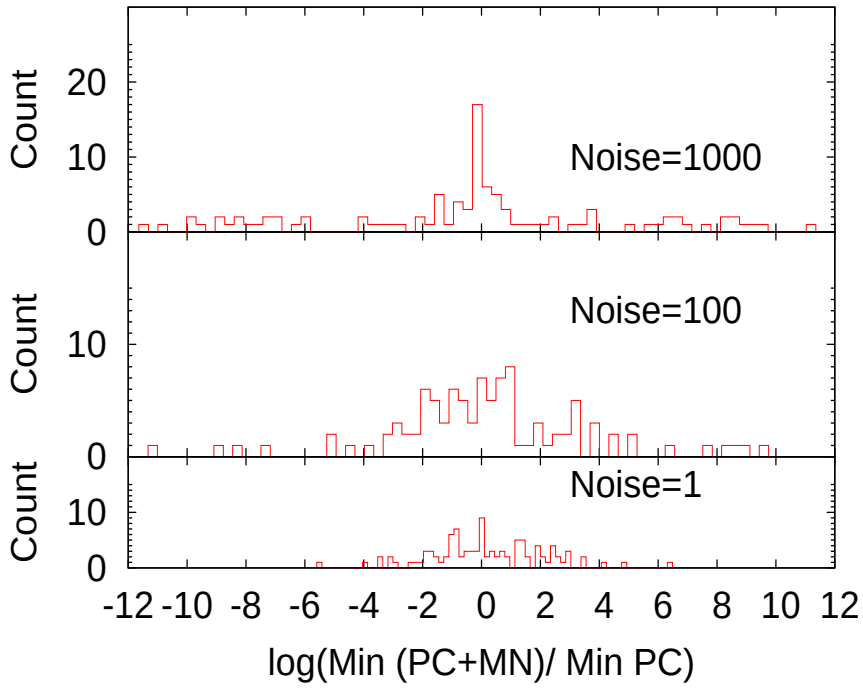
Figure 3.5: Performance of (a) MN vs. DET (b) PC vs. MN, and (c) PC+MN vs. PC, at three different noise levels ($\sigma^0 = 1, 100, 1000$), averaged over 100 different initial simplex states for Rosenbrock optimization.


Count
50
40
30
20
10
0
Noise=1000
Count
50
40
30
20
10
0
Noise=100
Count
10
0
Noise=1
-8
-6
-4
-2
0
2
4
6
8
log(Min MN/ Min DET)

(a)


Count
30
20
10
0
Noise=1000
Count
10
0
Noise=100
Count
10
0
Noise=1
-15
-13
-11
-9
-7
-5
-3
-1
1
3
5
log(Min PC/ Min MN)

(b)



(c)

Figure 3.6: Performance of (a) MN vs. DET (b) PC vs. MN, and (c) PC+MN vs. PC, at three different noise levels ($\sigma^0 = 1, 100, 1000$), averaged over 100 different initial simplex states for Powell function optimization.

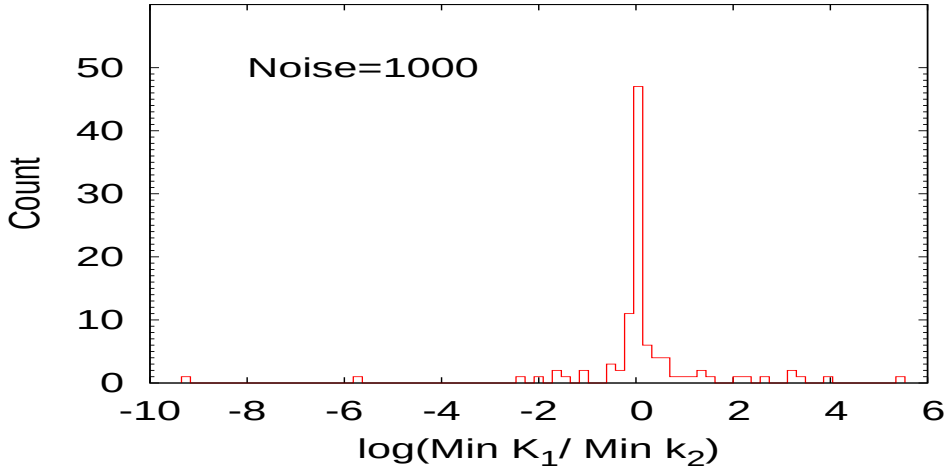


Figure 3.7: Performance of PC for K=1 vs K=2, at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

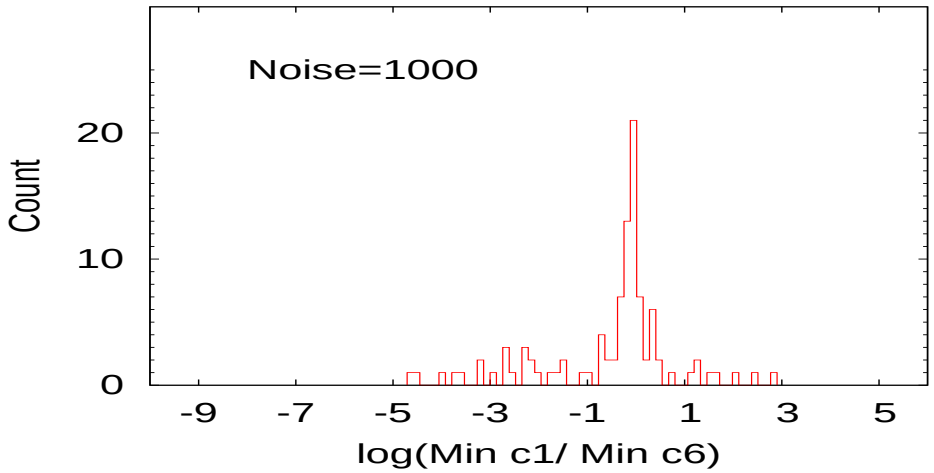


Figure 3.8: Performance of PC when considering error bar only in condition 1 (c1) compared to only condition 6 (c6), at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

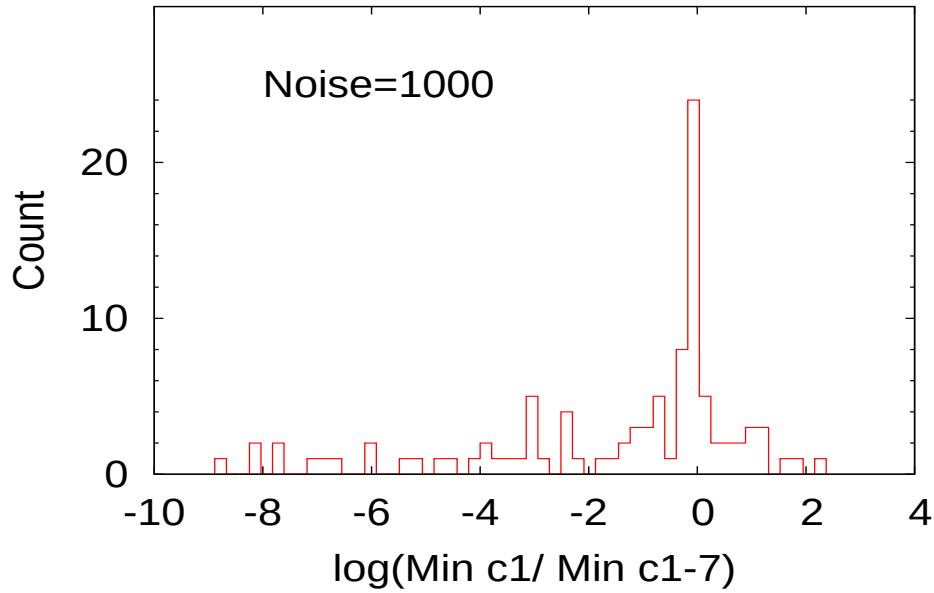


Figure 3.9: Performance of PC algorithm when considering error bar in condition 1 (c1) only and comparing with a strict implementation considering error bar in all conditions (c1-7), at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

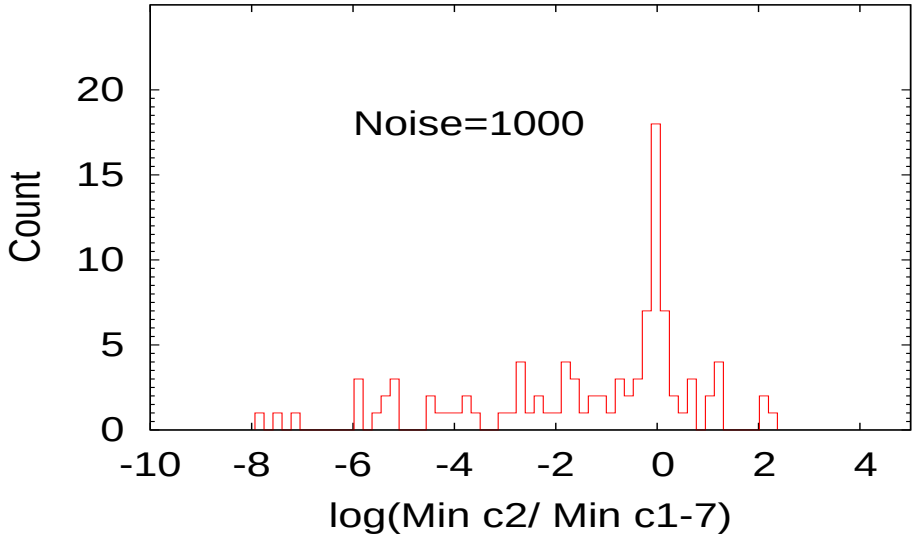


Figure 3.10: Performance of PC algorithm when considering error bar in condition 2 only and comparing with a strict implementation considering error bar in all conditions (c1-c7), at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

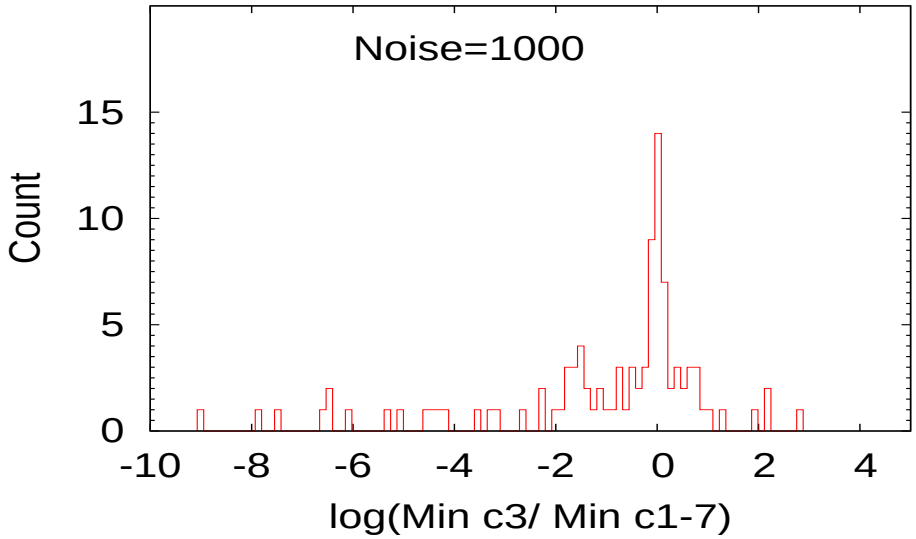


Figure 3.11: Performance of PC algorithm when considering error bar in condition 3 only (c3) and comparing with a strict implementation considering error bar in all conditions (c1-7), at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

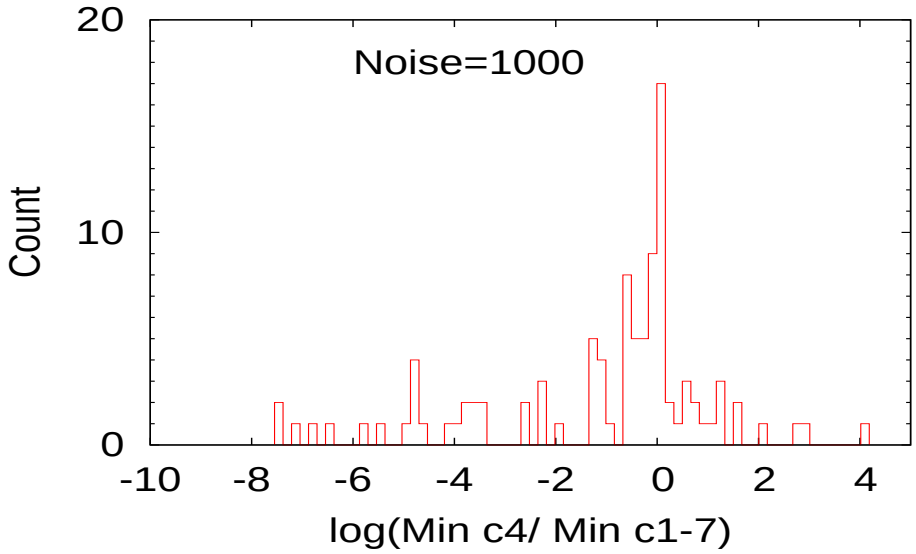


Figure 3.12: Performance of PC algorithm when considering error bar in condition 4 only (c4) and comparing with a strict implementation considering error bar in all conditions (c1-7), at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

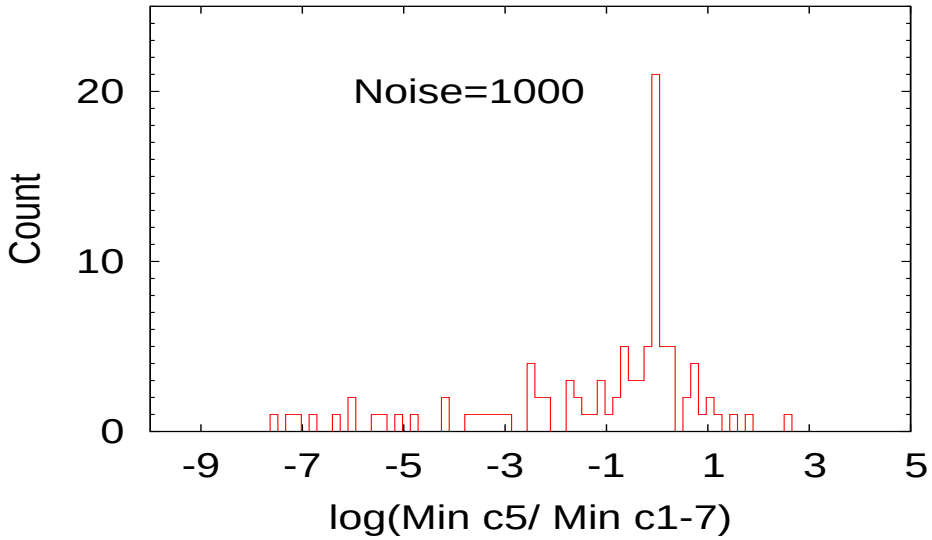


Figure 3.13: Performance of PC algorithm when considering error bar in condition 5 only (c5) and comparing with a strict implementation considering error bar in all conditions (c1-7), at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

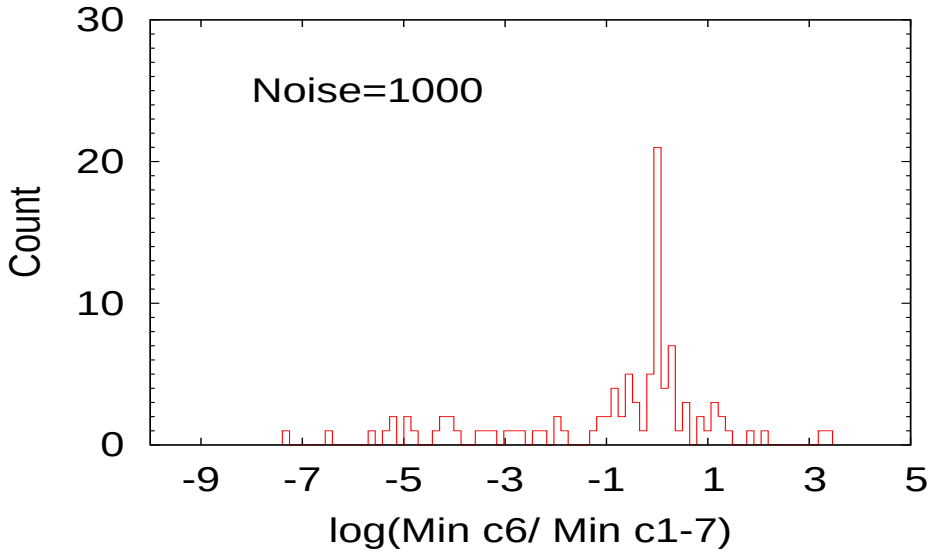


Figure 3.14: Performance of PC algorithm when considering error bar in condition 6 only (c6) and comparing with strict a implementation considering error bar in all conditions (c1-7), at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

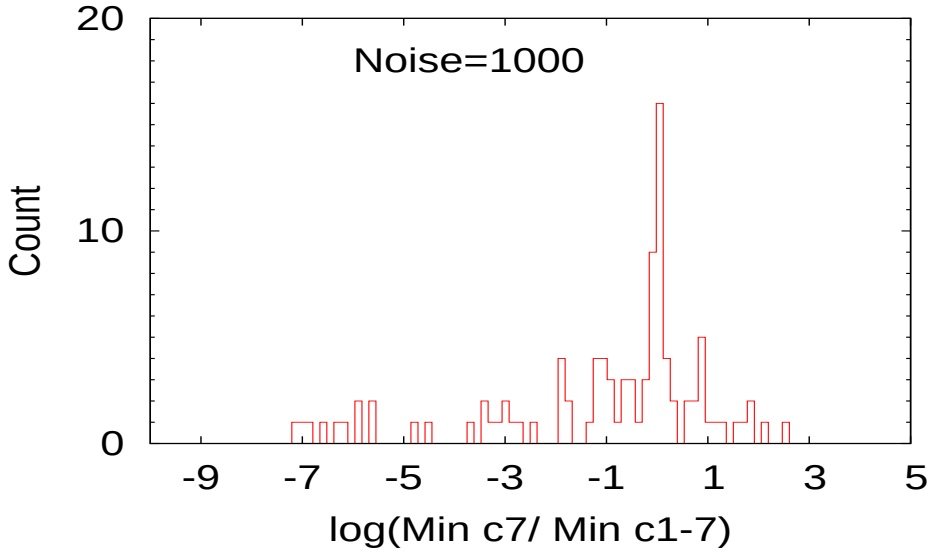


Figure 3.15: Performance of PC algorithm when considering error bar in condition 7 only (c7) and comparing with a strict implementation considering error bar in all conditions (c1-7), at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

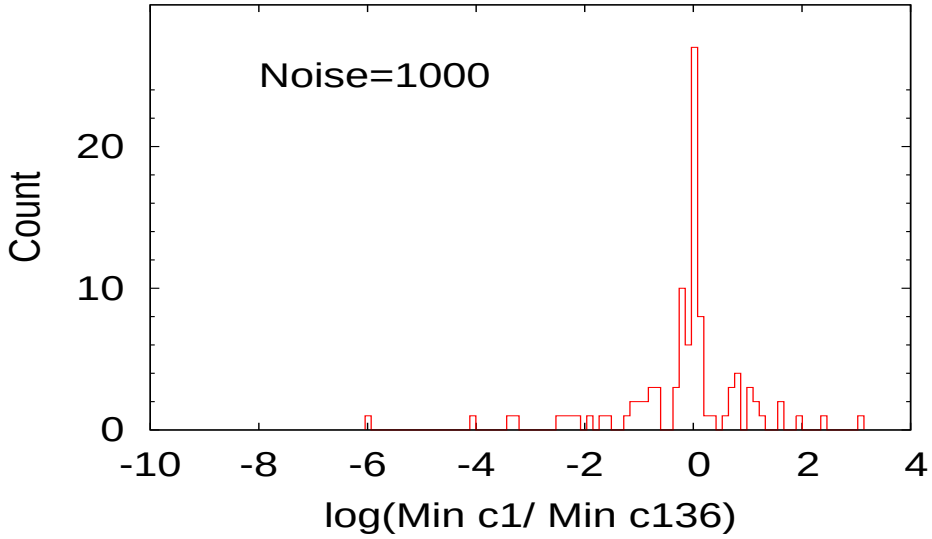


Figure 3.16: Performance of PC algorithm when considering error bar in condition 1 only and comparing with stricter implementation considering error bar in conditions 1, 3, 6, at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

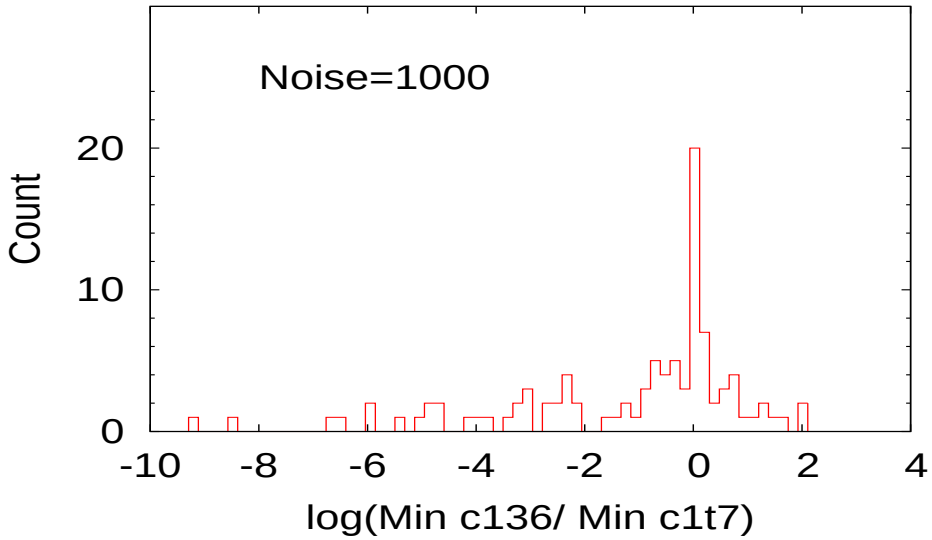


Figure 3.17: Performance of PC algorithm when considering error bar in condition 1, 3, 6 only and comparing with stricter implementation considering error bar in all conditions, at noise level $\sigma^0 = 1000$, averaged over 100 different initial simplex states for Rosenbrock optimization.

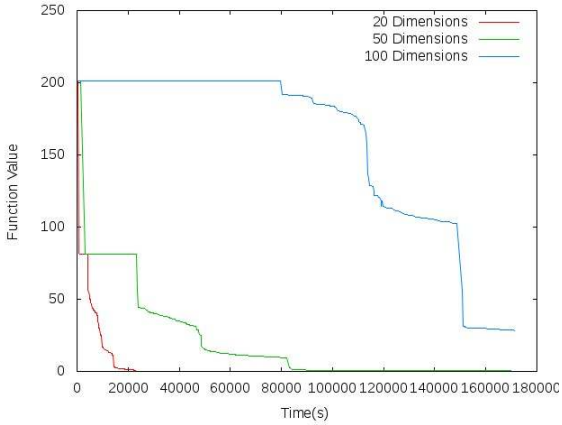
3.4 Scale Up

Table 3.3: Processor allocation for Rosenbrock optimization using MW framework.

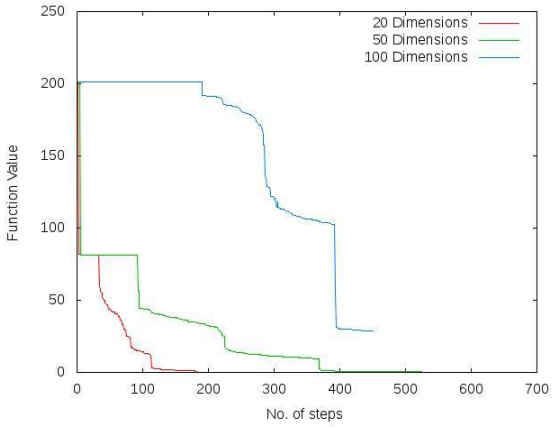
Dimensions (d)	No. of workers ($d+3$)	No. of servers ($d+3$)	No. of clients ($(d+3)N_s$	Total no of cores ($dN_s + 3N_s + 2d + 7$)
20	23	23	23	70
50	53	53	23	160
100	103	103	23	310

In many parameter optimization applications, scalability is a desired property, in which a growing parameter space can be handled in a graceful manner, without drastic performance degradation. We tested the MW design by optimizing the Rosenbrock function in d=20, 50, and 100 dimensions using 70, 160, and 310 processors, respectively, as shown in table 3.3. The Rosenbrock optimization using MW requires one master processor starting $d+3$ workers, and each server communicates with only one client executing the Rosenbrock function, such that the number of simulations running within each client-server framework is $N_s = 1$.

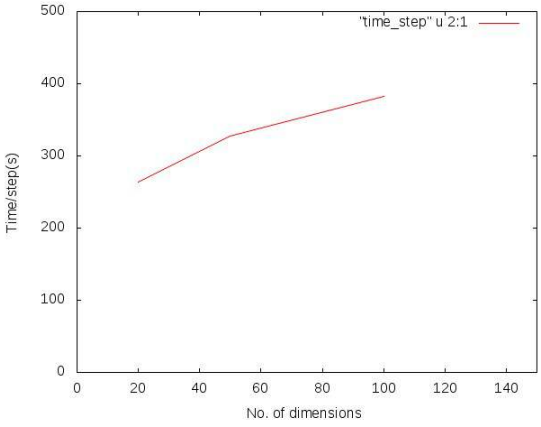
As the dimensionality of the problem increases, more steps are required (Figure 3.18b) by the simplex to converge to a minimum, as expected, hence requiring more time (Figure 3.18a). The increase in the time taken by the simplex to move a single step represents the performance degradation; this is minor (Figure 3.18c), and is attributed to the I/O at the simplex and vertex levels.



(a) function value vs time



(b) function value vs steps



(c) Time/step vs dimensions

Figure 3.18: MW Scale-up

3.5 Application

High performance computing is an indispensable tool for research in a domain like molecular dynamics. Molecular dynamics simulations are used to describe the chemical system using a numerical model. These simulations are extremely computationally intensive requiring significant resources. The numerical model is generally a potential that determines the physical or chemical properties of a system under consideration. The potential is further described by a list of parameters that characterize the model.

As an application of the MN, PC, PC+MN algorithms and the MW framework to a complex and realistic scientific application, we aimed to optimize the force field parameters $\{\Lambda_i\} = (\sigma, \epsilon, q_H)$ for TIP4P model of water [2], where ϵ and σ parameterize the Lennard-Jones interactions acting at the oxygen site, and q_H is the partial charge on the hydrogen atoms (Figure 3.19).

Thsese simulations consist of multiple phases though still considered to be a single simulation. An initial configuration is used to perform an MD equilibration in the NVT ensemble. The output of this simulation is used to perform a production run in the NVE ensemble. All these simulations require only an initial configuration, a fully parameterized potential and the description of the algorithm. The purpose of performing these simulations is to find the physical properties of the system like pressure, diffusion coefficient, temerature, density, etc. that are controlled by the parameter set Λ defining the potential V. TIP4P is among the most commonly used models for simulating liquid water, with well-studied properties, which makes it a good benchmark against which to compare our optimization algorithm. Note that TIP4P is already a very well optimized model; any errors in properties predicted by this model are primarily due to to assumptions in the functional form of the model

(Lennard-Jones and point-charge electrostatics) and the choice of model geometry, rather than errors in the model parameters. Our goal was not to improve the parameterization of the TIP4P model, but to use this well studied model as a convenient benchmark for the performance of our algorithms. The MW framework was used to perform a modified simplex optimization with both the PC and MN algorithms, while the client processes associated with each vertex performed a canonical ensemble (NVT) molecular dynamics (MD) simulation at 298 K in order to equilibrate the system, followed by a microcanonical ensemble (NVE) production run from which pair correlation functions and thermodynamic properties were evaluated.

The objective function $g(\Lambda)$ that we choose to optimize is the weighted sum of squares of six different residuals,

$$g(\Lambda) = \sum_{i=1}^{6} w_i^2 \frac{(p_i(\Lambda) - p_i^0)^2}{(p_i^0)^2} \tag{3.4}$$

where the $p_i(\Lambda)$ are the (noisy) equilibrium average properties obtained from simulation, p_i^0 are the experimental values of these properties, and w_i are the weights assigned to these properties. The weights were chosen subjectively to balance the level of error in each property. These properties included two thermodynamic properties (the average internal energy, $\langle U \rangle$ and average pressure, $\langle P \rangle$), one dynamic property (the self-diffusion coefficient, D), and three structural properties (obtained from the three radial distribution functions, g_{OO}, g_{OH}, and g_{HH}). All six properties were fit to experimental values [1, 73, 74]. The radial distribution functions were reduced to scalars by calculating the root mean square difference from the experimental curve, for example

$$p_{g(r)} = \left[\frac{1}{r_{max} - r_{min}} \int_{r_{min}}^{r_{max}} \left[g_{\mathrm{OO}}(r) - g_{\mathrm{OO}}^*(r) \right]^2 dr \right]^{1/2} \tag{3.5}$$

for g_{OO}, where $g^*_{OO}(r)$ is the experimental radial distribution function [1]. With this definition, the experimental (target) value for each $p_{g(r)}$ is zero. We specifically chose the diffusion coefficient and radial distribution functions as examples of the type of properties which are *not* typically fitted directly when developing molecular models, despite their importance, because they converge too slowly to be conveniently iterated over in a manual process.

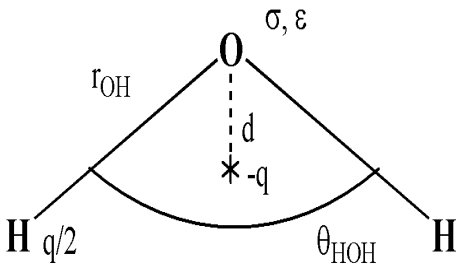
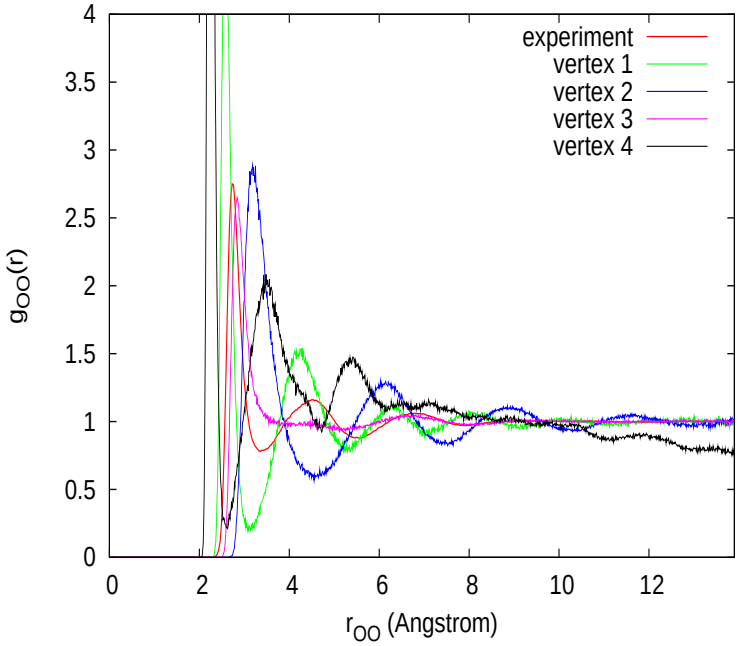
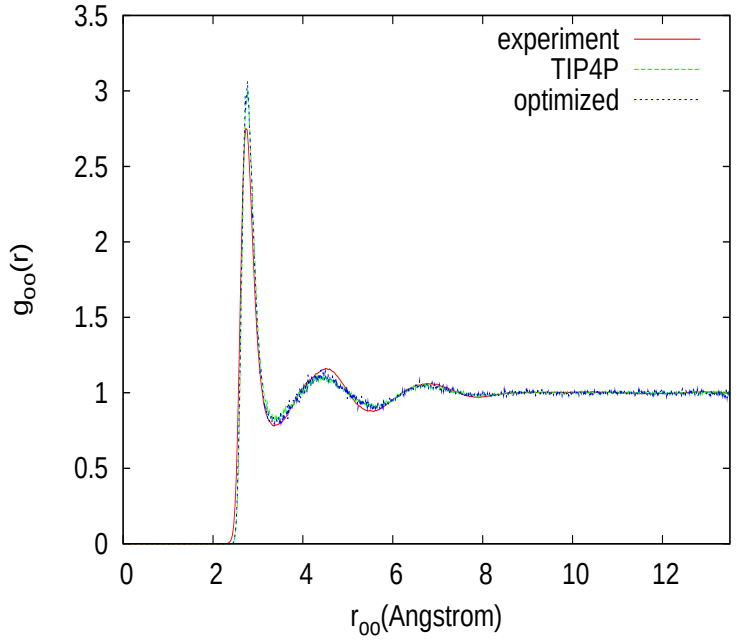


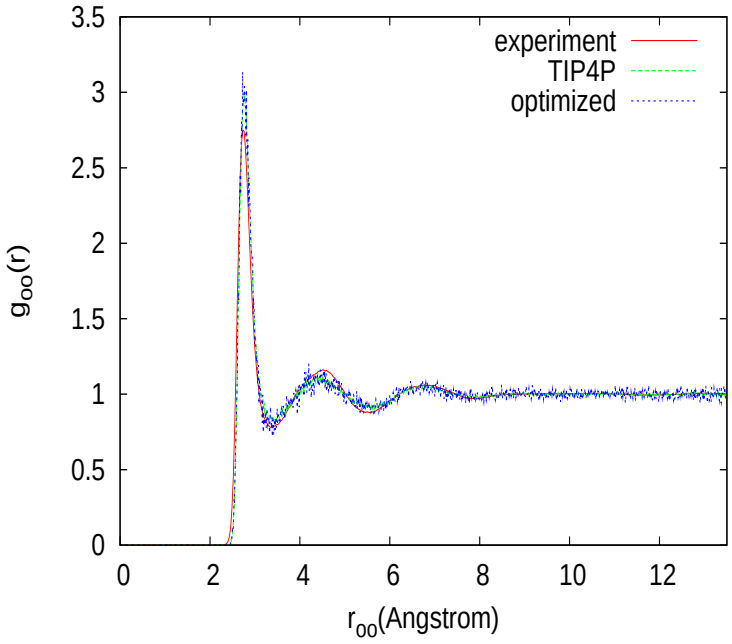
Figure 3.19: TIP4P water molecule model with parameters


experiment
vertex 1
vertex 2
vertex 3
vertex 4
$g_{OO}(r)$
r_{OO} (Angstrom)
0
0.5
1
1.5
2
2.5
3
3.5
4
0
2
4
6
8
10
12

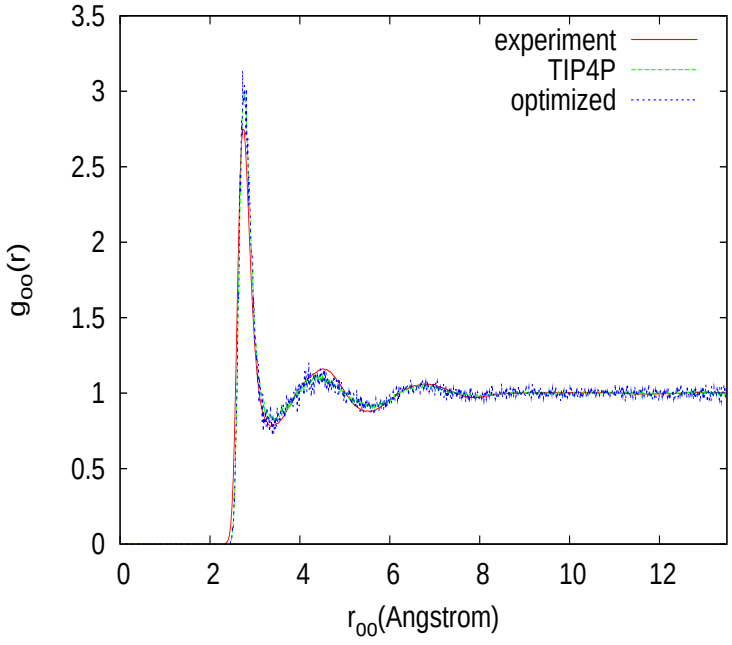
(a)


experiment
TIP4P
optimized
$g_{oo}(r)$
r_{oo}(Angstrom)
0
0.5
1
1.5
2
2.5
3
3.5
0
2
4
6
8
10
12

(b)



(c)



(d)

Figure 3.19: Oxygen-oxygen radial distribution functions (RDFs) for TIP4P water models with (a) non-optimal parameters, (b)parameters obtained using the MN algorithm, and (c) parameters obtained using the PC algorithm (d) parameters obtained using PC+MN algorithm, compared with RDFs obtained from experiment[1] and the standard TIP4P model[2].

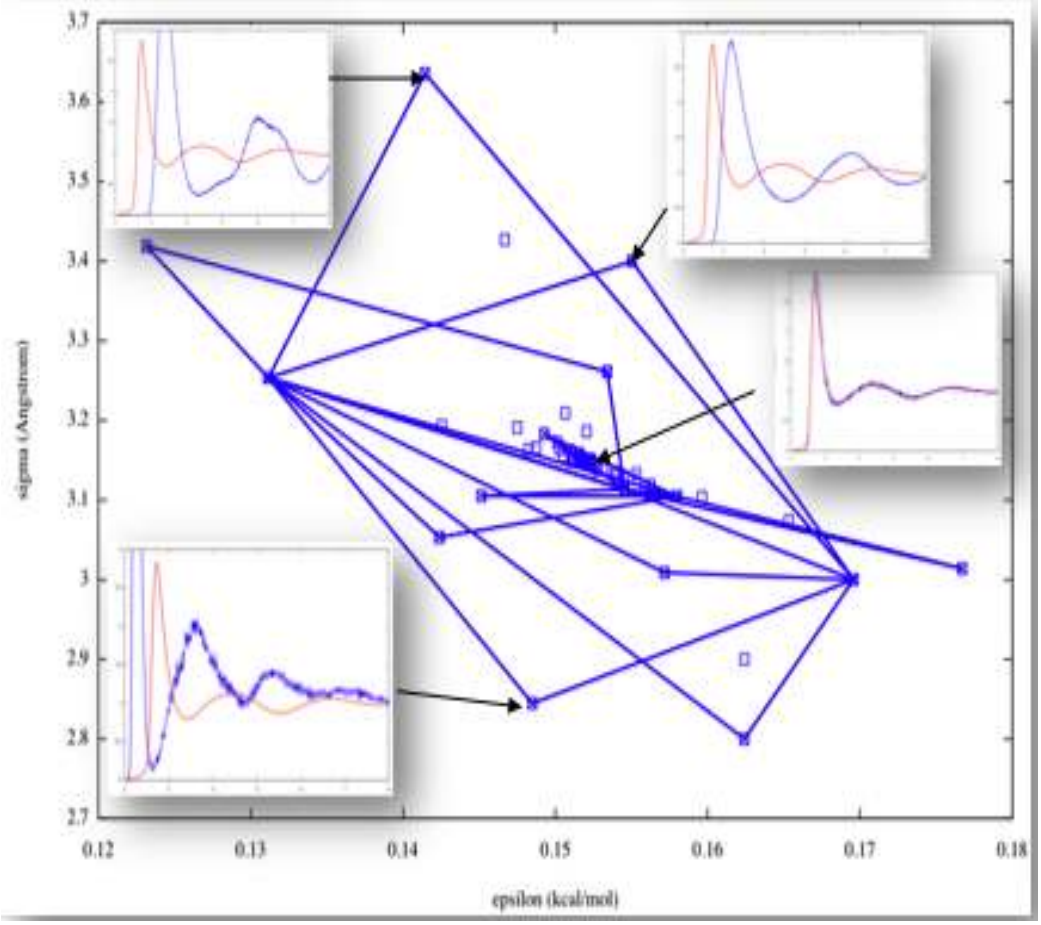


Figure 3.20: $g(r)$ curves for water model with parameters obtained from various stages of simplex optimization.

The simplex was initiated with parameter values (Table 3.4a) that gave poor and unphysical results, as illustrated by the initial $g_{\mathrm{OO}}(r)$ curves in Figure 3.20a. As the optimization process progressed, parameter values improved and resulted in better $g(r)$ curves as shown in Figure 3.20. Within 42 simplex steps, the MN optimization converged with values of ϵ = .1514 kcal/mol, σ = 3.150 Å, and q_H = 0.520$|e^-|$ (Table 3.4b). The PC optimization took 56 steps and converged with value of ϵ = .1470 kcal/mol, σ = 3.160 Å, and q_H = 0.523$|e^-|$ (Table 3.4c). The PC+MN algorithm required more than 62 steps and converged with values ϵ = .1470 kcal/mol, σ = 3.162 Å, and q_H = 0.522$|e^-|$ (Table 3.4d). These are all similar to the published

TIP4P parameters $\epsilon = .1550$ kcal/mol, $\sigma = 3.154$ Å, and $q_H = 0.520|e^-|$. Even more encouraging, the $g_{OO}(r)$ pair correlation function provides a slightly better fit to the experimental data with the MN, PC and PC+MN models than does the original TIP4P model, as shown in Figure 3.20b, Figure 3.19c, and Figure 3.19d, respectively.

The thermodynamic and dynamic properties are also well reproduced by the MN, PC and PC+MN models (Table 3.4). The average internal energy of the water is $\langle U \rangle = -41.69$ kJ/mol, -41.72 kJ/mol and -41.80 kJ/mol for the MN, PC and PC+MN parameterizations, respectively, while the experimental value is -41.5 kJ/-mol and TIP4P produces -41.8 kJ/mol [73]. All models give a pressure that differs substantially from the $\langle P \rangle = 1$ atm at the experimental density, but the MN, PC and PC+MN models give 212 atm, 368.5 atm and 266.8 atm respectively, compared to 373 for TIP4P. The diffusion coefficient improves from 3.29×10^{-5} cm^2/s for TIP4P to 3.00×10^{-5} cm^2/s with MN, 3.10×10^{-5} cm^2/s with PC and 3.01×10^{-5} cm^2/s with PC+MN, compared to the experimental value of 2.27×10^{-5} cm^2/s. Thus the MN, PC and PC+MN algorithms are capable of reproducing a model with properties equivalent to or better than the published TIP4P model.

Table 3.4: Numerical values of initial and final parameters obtained with MN, PC, and PC+MN algorithms.

(a) Initial parameters

$\epsilon_O(\frac{amu\mathring{A}^2}{df_s^2})$	$\sigma_O(\ \mathring{A})$	$q_H(\lvert e^- \rvert)$
7.1000×10^{-7}	3.0	0.54
6.4931×10^{-7}	3.40	0.45
5.4913×10^{-7}	3.25	0.52
6.8000×10^{-7}	2.80	0.60
5.4913×10^{-7}	3.25	0.60
6.8000×10^{-7}	2.90	0.65

(b) Final parameters obtained with MN algorithm

$\epsilon_O(\frac{amu\mathring{A}^2}{df_s^2})$	$\sigma_O(\ \mathring{A})$	$q_H(\lvert e^- \rvert)$
6.345×10^{-7}	3.153	0.5207
6.348×10^{-7}	3.153	0.5207
6.344×10^{-7}	3.153	0.5207
6.347×10^{-7}	3.153	0.5206
6.347×10^{-7}	3.153	0.5207
6.343×10^{-7}	3.153	0.5206

$\epsilon_O(\frac{amu\mathring{A}^2}{df_s^2})$	$\sigma_O(\ \mathring{A})$	$q_H(\lvert e^- \rvert)$
6.137×10^{-7}	3.169	0.5232
6.140×10^{-7}	3.168	0.5233
6.163×10^{-7}	3.166	0.5233
6.129×10^{-7}	3.168	0.5237
6.146×10^{-7}	3.167	0.5235
6.142×10^{-7}	3.168	0.5233

(c) Final parameters obtained using PC algorithm

$\epsilon_O(\frac{amu\mathring{A}^2}{df_s^2})$	$\sigma_O(\ \mathring{A})$	$q_H(\lvert e^- \rvert)$
6.231×10^{-7}	3.1610	0.5226
6.203×10^{-7}	3.163	0.5229
6.215×10^{-7}	3.163	0.5226
6.235×10^{-7}	3.160	0.5227
6.241×10^{-7}	3.161	0.5224
6.182×10^{-7}	3.164	0.5222

(d) Final parameters obtained using PC+MN algorithm

Table 3.4: Property (Pr) values(V) and error (E) : Diffusion constant (D), hydrogen-hydrogen (HH) $g(r)$, Oxygen-Hydrogen(OH) $g(r)$, Oxygen-Oxygen (OO) $g(r)$, Pressure (P) and Energy (E) as obtained using MN, PC, PC+MN compared with TIP4P and Experimantal data.

Pr	MN		PC		MN+PC		TIP4P	EXP
	V	E	V	E	V	E	V	V
D	3.0E-05	.50E-05	3.1E-05	.22E-05	5.06	2.23	1.30	1.35
HH	.0284	1.2E-05	.031	3.2E-05	.05	.0002	-	-
OH	.1015	6.1E-05	.102	.0001	.11	.0001	-	-
OO	.059	.000	.06	.0004	.09	0.0002	-	-
P	212.1	47.1	359.4	67.5	266.8	245	373	1
E	-41.69	.041	-41.68	.018	-41.80	.04	-41.80	-41.50

The average potential energy of the water model produced by the pair radial distribution function g_{oo} is also in agreement with the experimental values. The optimized parameters result in an average energy= -41.9 kJ/mol while the experimental value is -41.5 kJ/mol.

Chapter 4

Implementation

4.1 Hardware

The computing facility at Clemson provides the palmetto cluster of 1541 nodes and is the 80th fastest computer on the December 2010 Top 500 list [75]. Each node has dual processors and each processor has four cores for a total of 12328 compute cores.The Intel processors have a clock speed of 2.33GHz and have either 4 or 6 MB of cache. The Sun X2200 processors have a clock speed at 2.5 GHz and have 6 MB of cache. Each Intel node provides 12 or 16 GBytes of memory while Sun nodes have 16 GBytes of memory each. Also, each node has associated local disk, half of which can be used for temporary storage. The CPUs support both 32 and 64 bit applications. The cluster can achieve a peak performance of 85.04 TFlops using 1426 compute nodes. All the compute nodes run a 64-bit CentOS-5 Linux distribution with the 2.6.18-92.1.10.e15 version of the kernel. Palmetto provides a Myrinet Myri-10G [76] interconnection network between the nodes. The Myrinet 10G network is a high performance interconnect [77, 78] with low latency message passing and 1.2 GB/s of sustained network bandwidth . The Myri-10G network can be accessed in two ways:

using a low latency 9.8 Gbs IP network or with the Myrinet Express (MX) RDMA interface capable of achieving full line-rate bandwidth and a message passing latency of 2.3 μs. The message passing interface (MPI) [79] is the standard for communication between the nodes with in the parallel communication.

4.2 Software

The parameter optimization requires a number of inputs in order to define any particular optimization problem. These include the systems to be simulated, the properties to be calculated, and the cost function to be optimized. The purpose of this section is to identify how these various inputs are provided to the optimization program by the user. The user is responsible for providing the starting configuration for each simulation and the code (together with any other required input) used to perform the simulation. The initial points in parameter space needed to begin the optimization are provided by the user, while subsequent points in the parameter space are determined by the optimization algorithm. The scripts or codes used to perform property calculations are provided by the user, along with the target values for each property, and the weights used in the cost function. Details of how these values are specified by the user are described below.

Root

All user-specified information is provided via files that appear in a directory structure. The root of this directory structure is provided as an argument to the optimization program at runtime. All files that are used by the optimization program, or processes that it launches, are to be found with in this directory tree. Any two simultaneous instances of the optimization program should be run with distinct, non-overlapping directory trees, to avoid conflicts in output files. For the purpose of this document, we

will refer to the root of the directory tree as $OPTROOT. Figure 4.1 and Figure 4.2 shows the directory structure. The systems directory contains all the files required for the simulations. Also, the initial set of parameters is provided by the user via an *input* file stored in $OPTROOT directory. The first row in the *input* file provides the name of d parameters (separated by white space) to be optimized and the follwing $d + 3$ rows specify the coordinates (parameters) corresponding to $d + 1$ vertices of simplex.

Systems

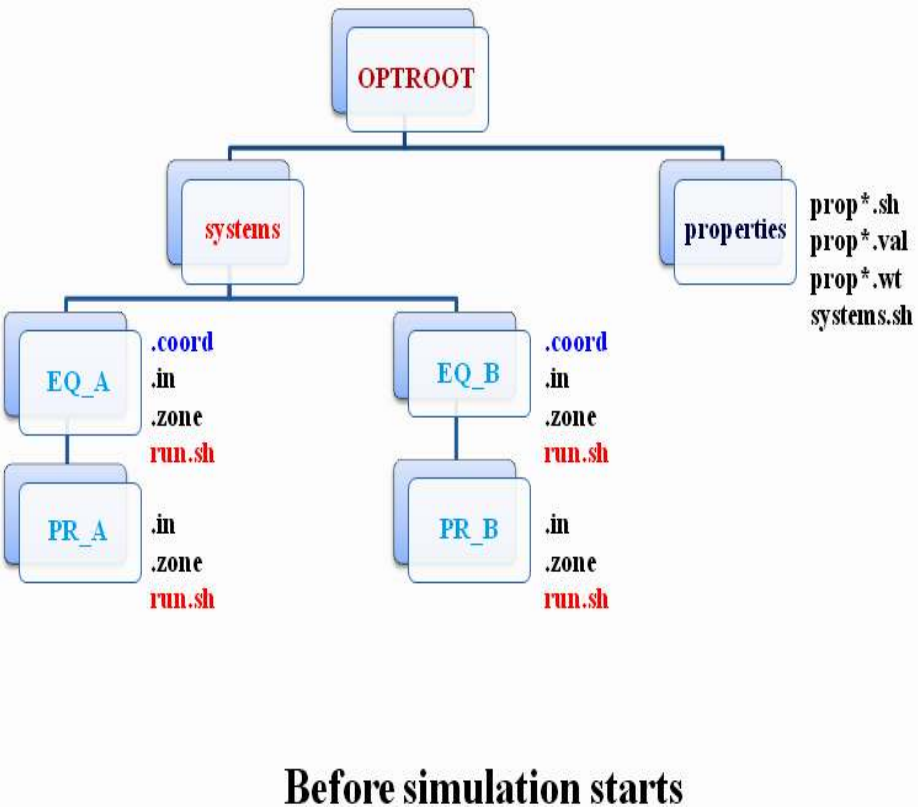


Figure 4.1: Directory structure created by user

At each point in parameter space, N_s systems must be simulated. Information about each of these systems is provided in a directory of the form $OPTROOT/systems/*sysname*. The subdirectory name *sysname* can be any valid UNIX directory

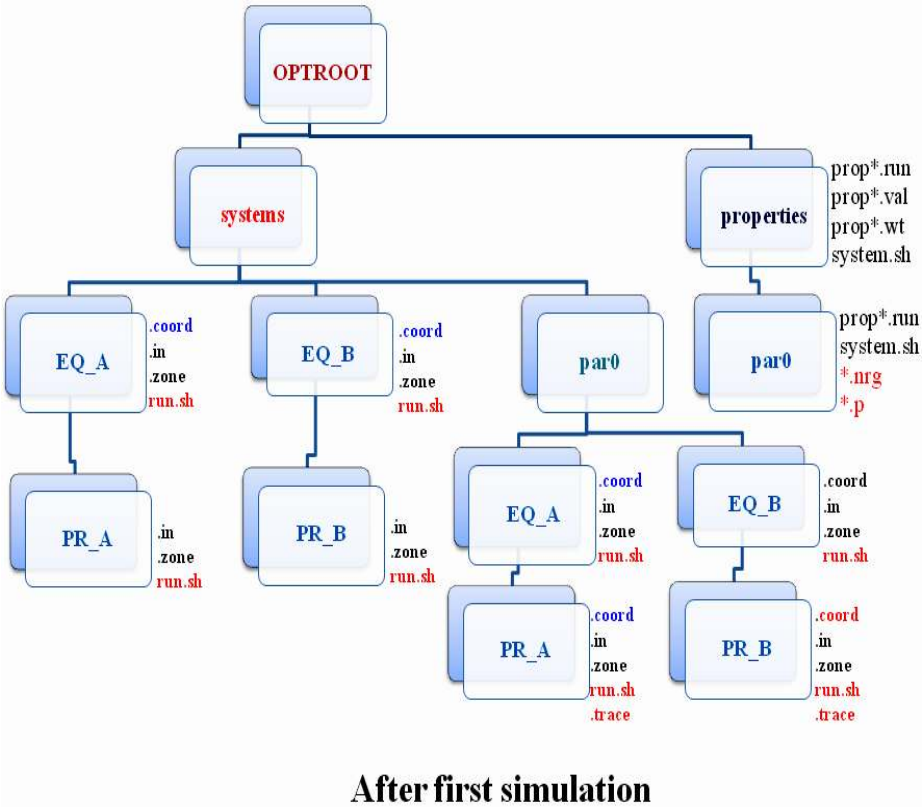


Figure 4.2: Directory structure after one simulation

name except that it can not match the regular expression par[0-9]*. Every subdirectory under $OPTROOT/systems that does not match the regular expression par[0-9]* is assumed to represent a system, and must contain all the files (described below) needed to run the system. Thus, no subdirectories beneath $OPTROOT/systems should be created that do not correspond to systems.

Starting Configurations

Each system (S_i) requires a starting configuration, (R_i). This configuration is provided by the user in one or more files in the $OPTROOT/systems/*sysname* directory. The format of the configuration files(s) is not specified by the optimization program, and should be appropriate for whatever program is being used to perform the simulation. Less than N_s different sysname subdirectories can be provided, if more than

one system uses the same starting configuration. **Simulation Phases**

In addition to a starting configuration, a simulation protocol T_i is also needed to fully specify a system. This simulation may consist of mutiple phases. The user provides self-consistent computational codes that perform the individual phases of the simulation, begining with the starting configuration file and resulting in the output files that will be needed to calculate the desired properties. The first phase of the simulation is performed by an executable file named $OPTROOT/systems/sysname/run.sh . Typically this will be a wrapper script that calls the appropriate executable for the first-phase calculation, in addition to doing any pre and post-processing. The user is responsible for making sure that all needed input files are provided, and that programs write files in the format that will be needed for the subsequent phases or property calculations.

If there is an (optional) second or later phase of simulation, the user provides another executable file as $OPTROOT/systems/*sysname/phasename*/run.sh. I.e., this program must appear in a subdirectory of the system directory. The directory name is not important, except that it must not match the regexp par[0-9] *; any other subdirectory of the configuration directory is assumed to be a second phase of the simulation. If there is more than one subdirectory, then more than one second-phase simulation will be performed with the same starting configuration. Thus it is important to ensure that no additional subdirectories are created that do not have run scripts and input files and that do not represent phases of simulation. The second phase of simulation will be initiated after completion of the first phase. Thus if the phase is implemented by the user through the use of a wrapper script, the wrapper script should not exit until the calculations are finished— for example by running the program in foreground rather than the background. Additional phases (3^{rd}, 4^{th},.....) are possible via nested subdirectories.

Property Values

The user is also responsible for providing the target values p_i^o for each property. These are provided in files with the name \$OPTROOT/properties/prop*.val. Each such file contains a single numerical value on the first line of the file, representing the target value for the property calculated by the corresponding \$OPTROOT/properties/prop*.sh script

The units of the target values should be same as those of the quantities calculated by the property scripts. These units need not be identified, but they must be consistent.

Property Weights

In order to evaluate the cost function g from the residuals in the property values (eq. 1.3), the weights,w_i, for each property must be provided. The user provides the value of w_i used in eq. 1.3, for each property p_i. When squared, this property is used as an inverse weight for the squared relative error in the property, as indicated in eq. 1.3. In practical terms, the w_i values can be considered to be a tolerance for relative error. For two properties p_i and p_i^1, if the value of w_i is twice as large as w_i^1, then a relative error of 10% in p_i will contribute the same amount to the cost function as a relative error of 5% in p_i^1 . Only the relative magnitute of the w_i values will affect the parameter optimization. The absolute scale of the w_i values is unimportant, and determines only the magnitude of the cost function.

Job submission

When the user scripts are placed in appropriate directories, the job is initiated by submitting a portable batch script (PBS) to the head node on the cluster from the \$OPTROOT directory. The job scheduler then interacts with the cluster Torque resource scheduler to determine when the available computing resources are granted to satisfy the jobs computing requirements. The submitted jobs may be queued for

several hours or even days. PBS requests the approriate number of processors on the cluster. The number of processors required for a system is calculated by the software using a wrapper script, which scans the directory structure and requests one processor for each *run.sh* script found.

Job Scheduling

We use our own scheduling for the MW implementation. PBS makes a copy of the machinefile ($PBS_NODEFILE) in the $OPTROOT directory, which contains the list of nodes (8 entries for each node) allocated to the job on the cluster. The job scheduler allocates one processor to the master and $d + 2$ processors to the workers from the machine file in order. Further, each worker starts a client-server job by allocating the required number of processors next available in the machinefile. When a worker is restarted by the master; it is restarted on the same processors using the machinefile. No new request is sent to the cluster for the processor allocation and scheduling is performed by our program. Also, new simulations due to restarted workers are carried out in a new directory under the $OPTROOT/systems directory.

4.3 Parallellization and Distribution

Analysis

We performed the analysis of the optimization algorithms to find which parts can be parallelized. The parallelization techniques can be categorized as *coarse-grain* and *fine-grained*. In a fine-grained implementation, very few data elements are assigned to one processor and information is exchanged frequently between the processors. On the contrary, a coarse-grained implementation has larger or sometimes even the entire data set implemented on a single processor and information exchange is rare. There are some components of the optimization process whose performance can be improved

remarkably by parallelization. The values of the cost function for any parameter set can be evaluated independently from the rest of the parameter sets. Furthermore for each parameter set, evaluating the individual cost functions for N_s different systems is usually a time-consuming part but each can be simulated independently of one another. Also, the property calculations are computationally inexpensive, but still they can be performed in parallel. Most practitioners make use of coarse-grained parallel structure by running multiple simulations as independent simulations on the different processors. The parallel communication is replaced by a manual process of calculating the property values from the output of each individual simulation. This problem generated a need for a formal parallel implementation. In the current implementation, overhead due to parallel communication is very low since communication costs are low while computation cost are high.

Distribution

In our research, variations of the downhill simplex algorithm are used to perform the optimization. This approach has the advantage of being easily parallelized at levels distinct from the cost function evaluation. In this implementation, each simplex vertex corresponds to a point in the parameter space. For each vertex, N_s simulations and N_M property calculations are performed. At any given time $d+1$ vertices are active performing $(d+1)N_s$ simulations and calculating at least $(d+1)N_M$ properties. At the highest level, the optimization algorithm defines $d+3$ vertices for a d dimensional optimization problem. This is implemented using one master process to implement the modified simplex and $d+3$ additional workers, one for each vertex (Figure 4.3). The master communicates with the wokers via MPI. There is no direct communication between the worker processes. The optimization algorithm navigates through the parameter space by performing the tranformation operations frequently by discarding one vertex and adding a new one as defined in the algorithm 1. Some

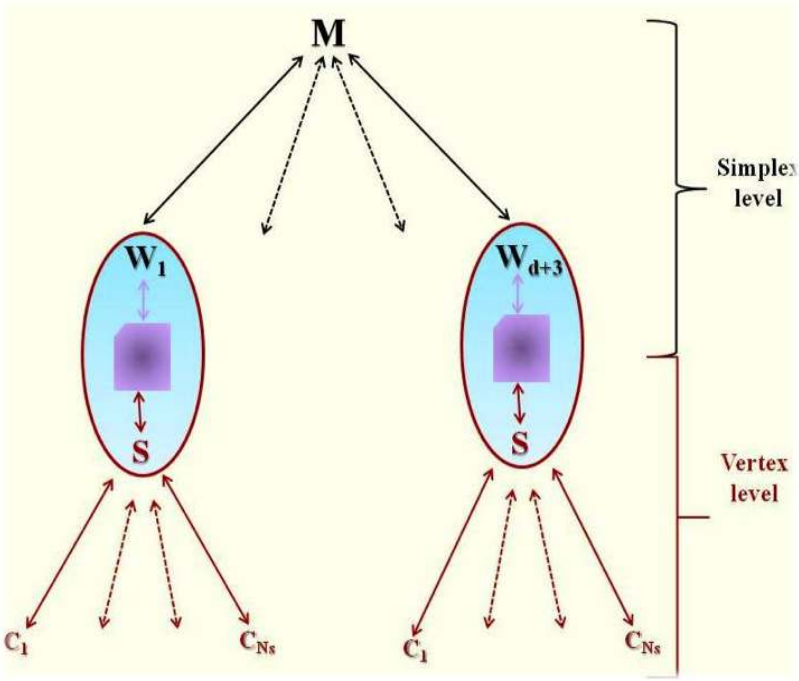


Figure 4.3: MW architecture

of the simplex operations require trial vertices. For example, sometimes the reflection of the simplex vertex generating the worst cost function is evaluted. If the reflected point is better than the original vertex than reflected vertex is kept; if not other operations are tested. Consequently, using more than $d+1$ worker processes, efficiency can be improved. We are using 2 additional vertices called the trial vertices in this parallel implementation. Using trial vertices has the advantage that the vertices are kept active until it is certain that they are no longer required. Thus we used total $d+3$ vertices to evaluate the cost function at $d+1$ workers and 2 trial vertices. Each of the workers perform N_s simulations that correspond to a single set of parameter values. These simulations are further performed in parallel using the client-server architecture. The client-server implementation at this level (vertex level) is different from the master-worker implementation at higher level (simplex level). Each vertex has one server process running and N_s client processing. Each client process maps on

to a single system. Also, each client runs a simulation starting with a configuration. These simulations can be done in multiple phases. The server process at the vertex level communicates with the client processes and coordinates the start and end of each simulation running on the client side. Server and client processes communicate via MPI at this level of parallelization. As in the simplex level, server communicates with each client but clients do not communicate with each other.

All the algorithmic decisions are made by the master at the simplex level and operations associated with individual simulations, or phases of a simulation are made by the clients. In this design, servers and workers are logically equivalent and there is one-to-one correspondance between them. Workers and servers participate in MPI communicatons upwards (between workers and master) and downwards (between server and clients), respectively. Workers communicate with their corresponding servers via file I/O.

Chapter 5

Conclusions and Discussion

5.1 Conclusions

Automated parameterization methods have the advantage of being faster, more efficient and more objective than optimization by hand, by reducing the computational effort, human involvement, and subjectivity. Several of the barriers to automated parallel parameterization, including the effects of noise on the objective function and the implementation of a parallel framework have been overcome in this work. We have developed three simplex based optimization algorithms to address the stochastic optimization bottleneck. We also developed a framework to automate the optimization process. We tested our algorithms on this frame work. This study also establishes that our model can be scaled up for large parameter spaces, and is well suited for cluster-based architectures.

The MN, PC and PC+MN methods presented here are suitable algorithms for advancing the simplex in the presence of a noisy objective function. All three algorithms attempt, with different criteria, to rationally decide how long to sample at each vertex before accepting a simplex move. Each stage in the parameterization process

terminates as soon as it can be determined that the parameters are non-optimal, but continues sampling as long as the parameters remain viable, rather than terminating at an arbitrary cutoff. All criteria used for advancing the simplex in the presence of a noisy objective function proved comparable or superior to methods used previously, with less requirement for problem-specific fine tuning. Our algorithms substantially outperform the standard deterministic simplex algorithm on the Rosenbrock test function, where the PC algorithm also outperforms MN, while PC+MN outperforms PC. The PC+MN algorithm provides results comparable to PC in accuracy, but can do so with fewer function evaluations.

We have applied the method to fit a potential for liquid water, using MD simulations and slowly convergent structural properties. In the application to fit a model for liquid water based on the TIP4P model, MN and PC and PC+MN algorithms result in a model that slightly improves upon the published model. This demonstrates that the algorithms can be used for real-world problems in parameter fitting. Our algorithms were quite robust in optimizing this molecular force field, even in the presence of a highly nonlinear objective function, and can be easily applied to other molecular modeling and more diverse parameter fitting applications.

One of the objectives of this work is to provide application developers an API with useful functionality compatible with a wide range of application codes. We successfully developed a programming model by enhacing the existing MW framework. The programming model enables user to develop the algorithmic features or the application code and embed in the framework without worrying about the details of the underlying framework. The following publications, posters, talks have resulted from this research:

Steven J. Stuart, **Dheeraj Chahal**, Sebastian Goasguen and Colin J. Trout, " Reparameterization of TIP4P water models with automated parallel stochastic opimization

methods" (*In preparation*).

Dheeraj Chahal, Steven J. Stuart, Sebastian Goasguen and Colin J. Trout, " Automated, Parallel Optimization algorithms for Stocastic Functions ", IPDPS Workshop on New Trends in Parallel Computing and Optimization, Anchorage, AK, USA, May 2011 (Accepted). [80]

Dheeraj Chahal, Steven J. Stuart, Sebastian Goasguen, Colin J. Trout, "Automated, Parallel Optimization of Stochastic Functions Using a Modified Simplex Algorithm" e-sciencew, pp.98-103, 2010 Sixth IEEE International Conference on e-Science Workshops, 2010. [81]

Colin Trout, Steven J. Stuart, **Dheeraj Chahal**, "Efficient Simplex Methods for Force Field Parameterization", SURP poster session, Clemson University, USA, July 2010. *poster* [82]

Pierce Robinson, Steve Stuart, **Dheeraj Chahal**, "Parameterization of Molecular Dynamics Simulations Using a Downhill Simplex Algorithm", SURP poster session, Clemson University, USA, July 2010. *poster* [83]

Dheeraj Chahal, Sebastien Goasguen, Steve Stuart, "Automated Parallel Parameterization Using Simplex Method", Workshop on Modeling Advanced Materials and Systems Biology: Building Capabilities and Collaborations for Cyber-Enabled Discovery, September 20-22,Clemson University, USA, 2010. *poster* [84]

5.2 Recommendations for Future Research

The suit of test problems to study the algorithms should be enlarged to include the test problem exhibiting diverse factors like: degree of difficulty, dimenstionality of system, response surface geometry. Also, one interesting study would be to test the algorithms under varying degree of dimensionality and different termination criterion.

We tested our algorithms for dimensionality $d = 2, 3, 4$ only, while the MW framework was tested for Rosenbrock function optimization with upto 100 dimensions.

Simplex has the potential of being used in conjunction with other algorithms with uphill movement capability. Many researchers have developed new algorihtms by combining simplex with other stochastic and global optimization algorithms. On the similar lines, the MN and PC algorithms can be tested with the these algorithms. For example, particle swarm optimization (PSO) suffers from the disadvantage of slow convergence in the refined search stages and has weak local convergence algorithm while the maxnoise, point-to-point and simplex in general lack the ability to converge to global minimum but converges quickly to a local minimum. An ability to use PSO with maxnoise and point-to-point may prove to be another step forward in the development of global stochastic algorithms.

From the results obtained we have proved that the modified simplex converges better than the classical NM simplex. Future research could include a theoretical proof of convergence of the modified simplex.

The new MW framework has considerable potential for improved speed-up. The current version of MW starts a client-server environment for each worker. Though worker and server are logically similar, they run on different cpus. A number of processors proportional to the dimension of the system under investigation can be saved by modifying the architecture and allowing to start the clients to be started directly from a worker without creating a server. Also, master and worker processes could easily share a single processor.

There is significant research going on in the development of framework for the distributed computing. Apache hadoop [85] and mapreduce [86] are popular open-source software packages for reliable, scalable, distributed computing. Studying new algoritms with these frameworks would be an interesting study.

Bibliography

[1] A. K. Soper. The radial distribution functions of water and ice from 220 to 673 k and at pressures up to 400 mpa. *Chemical Physics*, 258(2-3):121 – 137, 2000.

[2] William L. Jorgensen, Jayaraman Chandrasekhar, Jeffry D. Madura, Roger W. Impey, and Michael L. Klein. Comparison of simple potential functions for simulating liquid water. *The Journal of Chemical Physics*, 79(2):926–935, 1983.

[3] Andrew Lewis, David Abramson, and Tom Peachey. RSCS: A parallel simplex algorithm for the nimrod/o optimization toolset. *Proceedings of the ISPDC/HeteroPar'04, IEEE*, 2004.

[4] David Abramson, Tom Peachey, and Andrew Lewis. Model optimization and parameter estimation with nimrod/o. In *University of Reading*, 2006.

[5] Harold P. Benson. Deterministic algorithms for constrained concave minimization: A unified critical survey. *Naval Research Logistics (NRL)*, 43(6):765–795, 1996.

[6] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[7] I. Quesada and I.E. Grossmann. An lp/nlp based branch and bound algorithm for convex minlp optimization problems. *Computers & Chemical Engineering*, 16(10-11):937 – 947, 1992. An International Journal of Computer Applications in Chemical Engineering.

[8] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[9] Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM REVIEW 2003 Society for Industrial and Applied Mathematics*, 45(3), 2008.

[10] Virginia Torczon. On the convergence of pattern search algorithms. *SIAM J. on Optimization*, 7(1):1–25, 1997.

[11] Abramson D. A., A. Lewis, and T. Peachey. Nimrod/o: a tool for automatic design optimisation using parallel and distributed systems. In *Proceedings of the 4th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2000)*, pages 497–508. World Scientific Publishing Co, Singapore, 2000.

[12] Tsutomu Matsumoto, Hai Du, and Jonathan S. Lindsey. A parallel simplex search method for use with an automated chemistry workstation. *Chemometrics and Intelligent Laboratory Systems*, 62(2):129 – 147, 2002.

[13] Donghoon Lee and Matthew Wiswall. A parallel implementation of the simplex function minimization routine. *Computational Economics*, 30:171–187, 2007. 10.1007/s10614-007-9094-2.

[14] Louis Coetzee and Elizabeth C. Botha. The parallel downhill simplex algorithm for unconstrained optimisation. *Concurrency: Practice and Experience*, 10(2):121–137, 1998.

[15] J. E. Dennis, Jr., and Virginia Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1:448–474, 1991.

[16] Roland Faller, Heiko Schmitz, Oliver Biermann, and Florian Mller-Plathe. Automatic parameterization of force fields for liquids by simplex optimization. *J. Comput. Chem*, 20:100–9, 1998.

[17] Per-Ola Norrby and Tommy Liljefors. Automated molecular mechanics parameterization with simultaneous utilization of experimental and quantum mechanical data. *Journal of Computational Chemistry*, 19(10):1146–1166, 1998.

[18] A. Gaiddon, D. D. Knight, and C. Poloni. Multicriteria design optimization of a supersonic inlet based upon global missile performance. *Journal of Propulsion and Power*, 20(3):542–558, 2004.

[19] David G. Humphrey and James R. Wilson. A revised simplex serach procedure for stochastic simulation response surface optimization. *INFORMS Journal on Computing*, 12(4), Fall 2000.

[20] Andr I. Khuri and John A. Cornell. *Response surfaces : designs and analyses*. Marcel Dekker, New York, 1987.

[21] G. E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(1):pp. 1–45, 1951.

[22] Russell R. Barton and Jr. Ivey, John S. Nelder-mead simplex modifications for simulation optimization. *Management Science*, 42(7):954–973, 1996.

[23] Shu-Kai S. Fan and Erwie Zahara. Stochastic response surface optimization via an enhanced nelder-mead simplex search procedure. *Engineering Optimization*, 1:15–36, 2005.

[24] P. Gilmore and C. T. Kelley. An implicit filtering algorithm for optimization of functions with many local minima. *SIAM J. Optim*, 5:269–285, 1995.

[25] J.C. Spall. Adaptive stochastic approximation by the simultaneous perturbation method. *Automatic Control, IEEE Transactions on*, 45(10):1839 – 1853, 2000.

[26] J.C. Spall. Multivariatestochasticapproximationusingasimultaneousperturbationgradientapproximation. *Automatic Control, IEEE Transactions on*, 37:332–341, oct. 1992.

[27] Dirk V. Arnold and Hans-Georg Beyer. A comparison of evolution strategies with other direct search methods in the presence of noise. *Comput. Optim. Appl.*, 24(1):135–159, 2003.

[28] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998.

[29] Hans-Georg Beyer. Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):239 – 267, 2000.

[30] H.G. Neddermeijer, G.J. van Oortmarssen, N. Piersma, Rommert Dekker, and J.D.F. Habbema. Adaptive extensions of the nelder and mead simplex method for optimization of stochastic simulation models. Econometric Institute Report EI 2000-22/A, Erasmus University Rotterdam, Econometric Institute, 2010.

[31] Jeff Linderoth, Sanjeev Kulkarni, Jean-Pierre Goux, and Michael Yoder. An enabling framework for master-worker applications on the computational grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 43–50, Pittsburgh, PA, August 2000.

[32] The MetaNEOS Project. Metacomputing environments for optimization. Website, 2000. `http://www.mcs.anl.gov/metaos`.

[33] Qun Chen, Michael C. Ferris, and Jeff T. Linderoth. Fatcop 2.0: Advanced features in an opportunistic mixed integer programming solver. *Annals of Operations Research*, 2000, 1999.

[34] J.-P. Goux and Sven Leyffer. Mixed-integer nonlinear programming on metacomputing platform. Working Paper, 1999.

[35] J. Linderoth and S. Wright. A cutting plane code for stochastic programming on metacomputers. Invited presentation at the INFORMS National Meeting, November 1999.

[36] Jean-Pierre Goux, Jeff Linderoth, and Michael Yoder. Metacomputing and the master-worker paradigm. In *Preprint MCS/ANL-P792-0200, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne*, 2000.

[37] Jasper A. Vrugt, Breanndn Nuallin, Bruce A. Robinson, Willem Bouten, Stefan C. Dekker, and Peter M.A. Sloot. Application of parallel computing to stochastic parameter estimation in environmental models. *Computers and Geosciences*, 32(8):1139–1155, 2006.

[38] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[39] Hans-Paul Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[40] Anuradha Maria. *Genetic algorithm for multimodal continuous optimization problems*. PhD thesis, Norman, OK, USA, 1995.

[41] Fang Wang and Yuhui Qiu. Empirical study of hybrid particle swarm optimizers with the simplex method operator. In *Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, ISDA '05, pages 308–313, Washington, DC, USA, 2005. IEEE Computer Society.

[42] Fang Wang and Yuhui Qiu. Multimodal function optimizing by a new hybrid nonlinear simplex search and particle swarm algorithm. In Joao Gama, Rui Camacho, Pavel Brazdil, Alipio Jorge, and Luis Torgo, editors, *ECML*, volume 3720 of *Lecture Notes in Computer Science*, pages 759–766. Springer, 2005.

[43] Fang Wang, Yuhui Qiu, and Yun Bai. A new hybrid nm method and particle swarm algorithm for multimodal function optimization. In A. Fazel Famili, Joost N. Kok, José María Peña, Arno Siebes, and A. J. Feelders, editors, *IDA*, volume 3646 of *Lecture Notes in Computer Science*, pages 497–508. Springer, 2005.

[44] Fang Wang, Yuhui Qiu, and Naiqin Feng. Multi-model function optimization by a new hybrid nonlinear simplex search and particle swarm algorithm. In Lipo Wang, Ke Chen 0001, and Yew-Soon Ong, editors, *ICNC (3)*, volume 3612 of *Lecture Notes in Computer Science*, pages 562–565. Springer, 2005.

[45] Yuhui Qiu and Fang Wang. Improving particle swarm optimizer using the nonlinear simplex method at late stage. In Richard T. Hurley and Wenying Feng, editors, *IASSE*, pages 25–30. ISCA, 2005.

[46] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):pp. 671–680, 1983.

[47] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Readings in computer vision: issues, problems, principles, and paradigms. chapter Optimization by simulated annealing, pages 606–615. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[48] Fred Glover. Tabu search–part i. *INFORMS Journal On Computing*, 1(3):190–206, 1989.

[49] Fred Glover. Tabu search–part ii. *INFORMS Journal On Computing*, 2(1):4–32, 1990.

[50] Marco Dorigo and Gianni Di Caro. *The ant colony optimization meta-heuristic*, pages 11–32. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[51] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[52] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28 –39, 2006.

[53] Martin Middendorf, Frank Reischle, and Hartmut Schmeck. Information exchange in multi colony ant algorithms. In Jos Rolim, editor, *Parallel and Distributed Processing*, volume 1800 of *Lecture Notes in Computer Science*, pages 645–652. Springer Berlin / Heidelberg, 2000.

[54] Thomas Stutzle. Parallelization strategies for ant colony optimization. In *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, pages 722–731. Springer-Verlag, 1998.

[55] D.C. Montgomery and G.C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley and Sons, Inc, New York, 1994.

[56] K. Marti. *Stochastic Optimization Methods*. Springer, 2005.

[57] Brad L. Miller, Brad L. Miller, David E. Goldberg, and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[58] Brad L. Miller and David E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evol. Comput.*, 4:113–131, June 1996.

[59] Jack Lee and P. Wong. The effect of function noise on gp efficiency. In Xin Yao, editor, *Progress in Evolutionary Computation*, volume 956 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 1995.

[60] Sigrún Andradóttir. Accelerating the convergence of random search methods for discrete stochastic optimization. *ACM Trans. Model. Comput. Simul.*, 9:349–380, October 1999.

[61] Robert Michael Lewis, Virginia Torczon, Michael, and Michael W. Trosset. Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, 124:191–207, 2000.

[62] Rafael Rasa, Antonio Vidal, and Vctor Garca. *Parallel Optimization Methods Based on Direct Search*, volume 3991 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006.

[63] Patricia D. Hough, Tamara G. Kolda, and Virginia J. Torczon. Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing*, 23(1):134–156, 2001.

[64] Qiang Xiong and Arthur Jutan. Continuous optimization using a dynamic simplex method. *Chemical Engineering Science*, 58(16):3817 – 3828, 2003.

[65] Rachid Chelouah and Patrick Siarry. Genetic and nelder-mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. *European Journal of Operational Research*, 148(2):335–348, 2003.

[66] Rachid Chelouah and Patrick Siarry. A hybrid method combining continuous tabu search and nelder-mead simplex algorithms for the global optimization of multiminima functions. *European Journal of Operational Research*, 161(3):636 – 654, 2005.

[67] F. Herrera, M. Lozano, and D. Molina. Continuous scatter search: An analysis of the integration of some combination methods and improvement strategies. *European Journal of Operational Research*, 169(2):450 – 476, 2006. Feature Cluster on Scatter Search Methods for Optimization.

[68] Alejandro Karam, Gilles Caporossi, and Pierre Hansen. Arbitrary-norm hyperplane separation by variable neighbourhood search. *IMA Journal of Management Mathematics*, 18(2):173–189, April 2007.

[69] Nenad Mladenovic, Milan Drazic, Vera Kovacevic-Vujcic, and Mirjana Cangalovic. General variable neighborhood search for the continuous optimization. *European Journal of Operational Research*, 191(3):753 – 770, 2008.

[70] Joaqun Pacheco, Silvia Casado, and Laura Nuez. Use of vns and ts in classification: variable selection and determination of the linear discrimination function coefficients. *IMA Journal of Management Mathematics*, 18(2):191–206, April 2007.

[71] Edward J. Anderson, Michael, C. Ferris, and Himsworth These Methods. A direct search algorithm for optimization with noisy function evaluations. *SIAM J. Optim*, 11:837–857, 2000.

[72] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 1960.

[73] M. W. Mahoney and W. L. Jorgensen. A five-site model for liquid water and the reproduction of the density anomaly by rigid, nonpolarizable potential functions. *J. Chem. Phys.*, 112:8910–8922, 2000.

[74] D. Eisenberg and W. Kauzmann. *The structure and properties of water*. Oxford University Press, London, 1969.

[75] Top500 Supercomputing Sites. http://top500.org.

[76] Myricom Inc. http://www.myri.com.

[77] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 85–97, Denver, Colorado, June 1997.

[78] Brice Goglin. High-performance message passing over generic ethernet hardware with open-mx. *Parallel Computing*, In Press, Accepted Manuscript:–, 2010.

[79] Message passing interface forum , MPI: A message-passing interface standard. *Tech. Rep. UT-CS-94-230*, 1994.

[80] Dheeraj Chahal, Steven J. Stuart, Sebastian Goasguen, and Colin J. Trout. Automated, parallel optimization algorithms for stochastic functions. *IPDPS Workshops, IEEE International Conference on*, May, 2011.

[81] Dheeraj Chahal, Steven J. Stuart, Sebastian Goasguen, and Colin J. Trout. Automated, parallel optimization of stochastic functions using a modified simplex algorithm. *e-Science Workshops, IEEE International Conference on*, 0:98–103, 2010.

[82] Colin Trout, Steven J. Stuart, and Dheeraj Chahal. Efficient simplex methods for force field parameterization. *SURP poster session, Clemson University, USA*, July 2010.

[83] Pierce Robinson, Steve Stuart, and Dheeraj Chahal. Parameterization of molecular dynamics simulations using a downhill simplex algorithm. *SURP poster session, Clemson University, USA*, July 2010.

[84] Dheeraj Chahal, Sebastien Goasguen, and Steve Stuart. Automated parallel parameterization using simplex method. *Workshop on Modeling Advanced Materials and Systems Biology: Building Capabilities and Collaborations for Cyber-Enabled Discovery*, September 2010.

[85] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, 1 edition, June 2009.

[86] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.