# Global Optimization: Software and Applications

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Marina Schmidt

# Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

# Abstract

Mathematical models are a gateway into understanding theoretical and experimental. However, sometimes these models need certain parameters to be established in order to obtain the optimal behaviour or value. Optimization is a method to obtain certain parameters for optimal behaviour that may be a minimum (or maximum) result. Global optimization is a branch of optimization that takes a function and determines the global minimum for a given domain. Global optimizations can be come extremely challenging when the search space yields multiple local minima. Moreover, the complexity of the mathematical model and the consequent lengths of calculations tend to increase the amount of time required for the solver to evaluate the model. To address these challenges, two pieces of software were developed that aided the solver to optimize a black box problem. First software developed is Computefarm a distributed system that parallelizes the iteration step of a solver by distributing function evaluations to unused computers. The second software is a Optimization Database that prevents data from being lost as a result of an optimization failure, it is also used to monitor the optimization process and to store extra information on about the model.

In this thesis, both Computefarm and the Optimization Database were used in the context of two particular applications. The first is designing quantum error correction circuits. Quantum computers cannot rely on software to correct errors because of the quantum mechanical properties that allow non-deterministic behaviour in the quantum bits. This means the quantum bits can change states between $(0, 1)$ at any point in time. There are various ways to stabilize the quantum bits however, errors in the system of quantum bits and system to measure the states can occur. Therefore, error correction components are designed to correct for these different types of errors, to ensure a high fidelity of the overall circuit. A simulation of a quantum error correction circuit is used to determine the properties of components needed to minimize error. This simulation is optimized with the use of the Optimization database and Computefarm to obtain the properties need to achieve a low error rate. The second application is crystal structure prediction that minimizes the total energy of crystal lattice to obtain its stability. From doing this stable structures of Carbon and silicon dioxide is obtained by using Computefarm and the Optimization Database obtains and store various metastable to stable structures. The use of the database then is used to obtain the top $N$ metastable structures of Carbon and Silicon Dioxide for further research to be done on.

# Acknowledgements

I would like to say thank you to my supervisor, Dr.Raymond Spiteri, for the support and push when I need it.

To my parents and family

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| SCUBA | Self Contained Underwater Breathing Apparatus |
| LOF | List of Figures |
| LOT | List of Tables |

# Chapter 1

# Introduction

Optimization is a process to obtain a minimal (or maximal) result for a defined problem. Problems can be a simple function like $f(x) = x^2$ to multiple functions that make up a complex model like determining the water saturation in a fuel cell. By optimizing these types of problems, known as objective functions, parameters for a given objective function is solved to minimize the result. This can lead to new discoveries in research and push forward new technologies and ideas of understanding. Optimization can be local or global. Local optimization searches in a given neighbourhood around a provided candidate solution. This obtains a local minimum value in the neighbourhood of the domain based on satisfying converging properties. In the case of searching for global minimum where the local minimum is not guaranteed to be a global minimum then a global solver is needed. A global optimization will search the whole domain unlike the local solver that searches a neighbourhood of the domain. By doing this the global optimization process determines a global minimum in certain amount of time. There are two types of global optimization: deterministic and stochastic [12]. These types of global optimizations have found global solutions for the applications chemical equilibrium, nuclear reactors, curve fitting, vehicle design and cost [14], and many others.

Deterministic algorithms guarantee a global minimum however, they take a large amount of time to do so because they search the whole domain. In the worst case, these algorithms have time complexities of exponential time. In the case where large amounts time is not practical, stochastic algorithms are used.

Stochastic algorithms cannot guarantee a global minimum because of the adaptive random searches they use. However, they can determine a good candidate minimum in a specified amount of time. Researches tend to use these algorithms if the exact global minimum is not needed to solve the objective function. There are also various stochastic algorithms that may preform better on specific type objective functions. Some examples of algorithms are Particle Swarm Optimization [11], Genetic Algorithm [3], Simulated Annealing [3].

Various types of objective functions lead to various challenges for the global solver. One type of challenge is premature local convergence, this happens when a global solver gets stuck on exploring a local minimum. This wastes time on not finding a global minimum and can lead to longer optimization time for deterministic algorithms or not a good global candidate solution for stochastic algorithms. Objective functions that have the potential of premature local convergence is non-convex functions where local minima are not guaranteed to be the global minimum. Thus a solver gets stuck in a local minimum making it difficult to find a global

minimum value. Other challenges is when failure occurs in the objective function. The optimization process is challenged with graceful shutdown that may lead to restarting from the beginning or re-evaluating the objective function in hopes a failure does not occur. A failure can occur when a resource contention happens. Resource contention occurs when the demand for resources are higher than the computer can provide. When multiple objective functions are running simultaneously the demand for resource power increase by a factor of the number of objective functions running, contention then occurs. Depending on the machine and the algorithms policy this can lead to serial optimization or a failure. If it is ran serially then this defeats the purpose of speeding up the optimization process. In the latter case, the optimization process will be restarted. Another challenge of global optimizations is when the objective function needs to be monitored to determine if further action is needed or obtain extra information. The monitoring aspect of an objective function is when the user has flexibility to change various properties of the function. This can change the sensitivity of the optimization process to obtain a global minimum faster or slower. Other situations is when various external information is desired by the objective function that the user may want to know

- specific properties of the model,
- post processing of the data, or
- knowing the top $N$ solutions.

My contributions to solving these challenges are the software Computefarm and the Optimization Database. Computefarm is a distributed system that utilizes unused computer resources on various client computers to run multiple function evaluations. As a result, it can speed up the iteration process of a given solver, and thereby speed up the search for the global minimum in a black-box problem. It is also fault tolerant to failure of a farmed computer allowing the global optimization process to continue without needing to be restarted.

The Optimization Database is a flexible database that allows the model or solver to store the result and extra information that may pertain to the simulation or post processing of the data. With this information the user can i) determine if the solver is stuck at a local minimum, ii) initiate other solvers based on currently stored data, or iii) use the data to further analyse the model.

In this thesis we apply Computefarm and Optimization Database to two applications quantum error correction circuit design and crystal structure prediction. In the first problem, a circuit is designed to correct for errors in quantum processor systems. Unlike transistor based computers, quantum computers cannot be controlled using a software-based design. Instead, they are controlled directly from a circuit component. This makes the process of manufacturing circuits and testing of desired reliability quite costly. In light of this, several models have been designed to simulate a quantum error correction circuit to determine the effect of error correction on a given $n$-qubit system. To ensure high reliability, also known as feasibility, of the circuit design, the circuit parameters are optimized such that the model returns a desired feasibility. In this application the desired feasibility is 99.99% for a encryption, decryption circuit [4, 6]. This is the highest modelled feasibility needed due to external noise when manufactured. This feasibility value has been obtained

for the 3-qubit system [17]. In this thesis, the 4-qubit system is reached using global optimization algorithms and the software applications.

Crystal structure prediction has been used in the past decade to approximate the most stable structure of a compound at a particular temperature and pressure environment. Because there is a large variety of lattice positions for a given compound, testing every possible lattice structure at a desired temperatures and pressures can be taxing. Thus researchers use Ab Initio structure codes to calculate properties of the given lattice in hopes of discovering a new structure for a given compound. One particular property that Ab Initio codes return is the total energy of the provided lattice structure. This energy represents the stability of the structure in the given environment. The lower the energy, the more stable the structure and the potential of having interesting properties. For example, finding a structure harder than diamond seems impossible experimentally however, by globally optimizing a specific compound in a given environment a new stable structure can be discovered that may be harder than diamond. Another region of interest for structure prediction is the development of crystals for electronics. This is because crystals have nice resonating frequencies and for this are used to build signal timing components in many devices like wrist watches and computer processor chips. Global optimization is needed to find the most stable structures for the crystal compound to further determine properties like the optimal resonant frequency. In this thesis the diatomic structure of silicon dioxide is optimized to generate the most stable lattice structures of a quartz crystal to further understand the variations of each structure and their crystal properties.

In this thesis, Chapter 2 gives the background on global optimization algorithms and properties objective functions present to optimization process. In Chapter 3 documents the software developed, Computefarm and the Optimization Database. Chapter 4 describes the two applications, quantum error correction circuit design and crystal structure prediction of silicon dioxide and how the software was used to aid in solving these problems. The final Chapter 5 summarizes the results obtained for the two applications and benefits the software had to solving the applications.

# Chapter 2

# Global Optimization

Global optimization is widely used method in engineering, chemistry, economics and etc, to obtain the extreme minimum (or maximum) value. By formulating real world problems to obtain the global minimum in the form of

$$\begin{aligned} x^* &= \arg\min_{x \epsilon D} f(x) \\ f &: D \to \mathbb{R} \\ D = x, x \epsilon \mathbb{R}^N, l_n \leq x_n \leq u_n, n &= 1, \ldots, N \end{aligned} \tag{2.1}$$

where $f(x)$ is the objective function, x is a $N$-dimensional vector between the lower bound $l_n$ and upper bound $u_n$ for the $n$th variable in the domain $D$ and $x^*$ denotes the global minimum of the objective function in the given domain $D$. This form of global optimization 2.1 represents an *unconstrained global optimization* expression; to add constraints the form would include

$$\begin{aligned} &subject\ to \\ g(x) &\leq 0, \\ h(x) &= 0 \end{aligned} \tag{2.2}$$

this for is known as *constraint global optimization.*

In the 1950's exhaustive searches like the Simplex algorithm [12] used this form 2.1 to obtain the global minimum by searching over ever possible point in the domain. This method of searching did guarantee the global minimum over enough time to search every possible combination. This became non-practical to users as objective functions became more complex and domains larger. In the most recent years, research has looked into the characteristics of the objective function, the constraints and algorithms to develop more efficient methods to obtain the global minimum.

## Objective Functions

An objective function is the computation that the global optimization is solving. This function can contain a single function or multiple functions to represent a model of a real world problem. These functions have

multiple characteristics that are used to determine the difficulty of the global optimization process in the given domain, some examples are

- continuity,
- smoothness,
- convexity,
- differentiable, and
- computational time.

*Continuity* is defined as

$$\lim_{x \to b} f(x) = f(b) \tag{2.3}$$

where $x$ and $b$ are independent points of each other. If Equation 2.3 is satisfied for every point in the set $b$ then the function is *continuous*, otherwise it is *discontinuous*. If the set is distinct and unconnected then the function is *discrete*, typically integers or whole numbers. In global optimization knowing the continuity determines which algorithm to use for the global optimization process. For discrete functions specific algorithms have been developed to handle the space constraints. Some examples of these algorithms are, Discrete Particle Swarm Optimization [10] and Branch and Bound algorithm [12], where they round up numbers to nearest set value. In the case of discontinuous objective functions, the space or constraints are modified to make the function either continuous or seem continuous to the solver. Examples of these modifications are

- return penalizing result when a gap or asymptotic region is explored,
- constrain the space to contain the continuous region of the function,
- add constraints to avoid gaps or asymptotic regions.

*Smoothness* is defined is the highest order of the continuous derivatives the function can achieve. The higher the order of the continuous derivative the smoother the function is. For global optimization the smoother the function the easier it is to converge on a minimum. In the case of non-smooth functions it becomes difficult to converge on global minimum because inconsistency of the landscape. One example of a non-smooth function are probabilistic functions that cannot not guarantee the same result when repeated. In this case the objective function can return the average of multiple runs of the functions or a hybrid algorithm that uses a local solver to converge to points in the sub-region to smooth out sub-regions, for example hybridizing particle swarm optimization [10].

*Convexity* is defined as the curvature of the function. A function is *convex* if for any $x_1, x_2 \epsilon X$ that follows

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2),\ 0 \leq \lambda \leq 1. \tag{2.4}$$

If the Equation 2.4 does not hold the function is *non-convex*. When a function is convex all local minima are global minima therefore only a local optimizer is needed to solve the problem. However, when the function is non-convex then not every local minima is a global minimum therefore the whole domain needs to be search to guarantee one of local minima is a global minimum, this is shown in Figure 2.1.

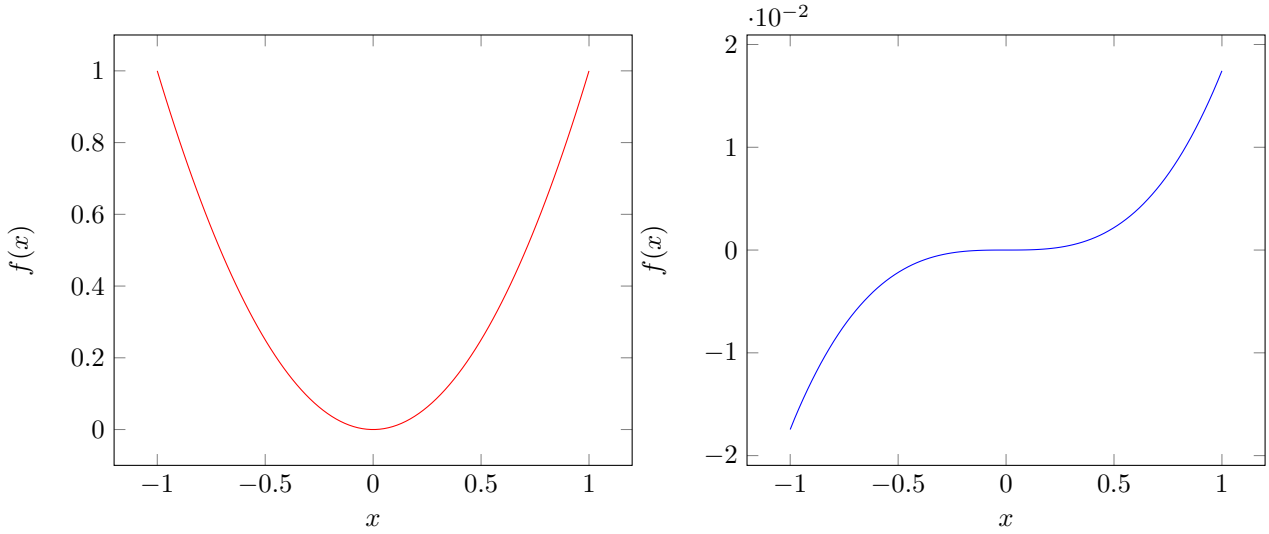



**Figure 2.1:** Comparison between a convex and non-convex function.

In the case of a non-convex function a global optimization method is used to solve the problem.

*Differentiable* is defined if a functions derivative exists. If a functions derivative does exist, then it can be used to aid global optimization to converge to a solution faster. The solver uses the derivative to determine the *gradient* of function. The gradient is a vector that gives a direction to slope of the surface. If the gradient moves between negative to positive slope then the solver is approaching a minimum and vice versa for maximum. When the gradient is

$$\bigtriangledown f(x) = 0, \tag{2.5}$$

then a minimum or maximum is obtained. Gradient methods are faster at converging to minima values and are used in some global optimization algorithms like Multi-Level Single Linkage algorithm [12].

*computational time* is the time it takes to evaluate the objective function at a given point. In global optimization this is a factor in optimization time and if concurrency is needed. To speed up the optimization time the objective function can be done in parallel if possible or the use of a parallel algorithm, for example parallel particle swarm optimization [8].

Objective functions are the core of global optimization that can aid in the optimization process or make it challenging. As discussed in this section certain characteristics are also used to determine which algorithm to use. In the situation when then objective function's characteristics are unknown or cannot be provided, a *black box global optimization* solver is used. Black box refers to a unknown objective function that in takes

in input information then returns a value. Theses are commonly known as *derivative-free* algorithms like Genetic Algorithm [3], Differential Evolution [3], Particle Swarm Optimization [11], etc.

## Constraints

Constraints are conditions placed by the user or the algorithm that must satisfy a given condition. For user defined constraints they can be implicit or explicit constraints that need to satisfy a condition. *Explicit constraints* satisfy an equality or inequality constraints, typically:

$$g_i(x) = 0, h_j(x) \leq 0 \tag{2.6}$$

where $i$th and $j$th constraints for constraint global optimization algorithms. In the event the constraints are not satisfied the algorithm will apply a penalty to the evaluated position. Penalties vary based on algorithms they can be:

- function set by the user,
- increase (or decrease) the returned value by a factor,
- increase search basin for local search region.

*implicit constraints* are constraints set within the objective function. This occurs when algorithms do not provide the option to set explicit constraints. In this case the user will then implement a constraint in the objective function that will return a penalized value. In some cases the algorithm will combine explicit constraints to the objective function, an algorithm that incorporates this is the *augmented Lagrangian method* [13], where explicit constraints are combined with the objective function to create a sub-problem that is optimized by a local solver. When constraints are not met the sub-problem will penalize the value and the process will be repeated until a solution is obtained.

Some algorithm place constraints internally on the objective function to aid solving the problem. An example of this is the $\alpha$-Branch and Bound algorithm developed by [2], where internal constraints are placed on the lower and upper bounds of the objective function to segment sections that make the function convex. This is a form of *convex relaxation*, where certain manipulations to non-convex function changes the function to be convex. In this situation the bounds were changed until the function is convex to then converge on a local minimum in that region. The algorithm then repeats for other regions and compares each local minimum to determine the global minimum.

Other internal constraints are seen as termination conditions, for example

- number of evaluations,
- time limit,
- function tolerance,

- position tolerance, and
- no change in current best solution.

Any of the above conditions set in an algorithm that has to be satisfied before termination of the global optimization process.

# Algorithms

In the past recent years global optimizations algorithms are broken into two main types, stochastic and deterministic.

## Deterministic

Deterministic algorithms guarantee finding the global minima by searching the whole domain with tight convergence properties [14]. In 1969 the branch and bound algorithm [12] was one of the most well known algorithms for solving complex models like the travelling salesman problem [12]. Other deterministic algorithms include interval optimization [1], algebraic technique [16], and DIRECT [9]. One weakness of deterministic algorithms is they can take large amount of time to find the global minimum that is not practical to the user.

## Stochastic

Stochastic algorithms were developed in the seventies to use adaptive random methods to obtain a feasibly close global minimum solution in a reasonable amount of time to the user. These algorithms, unlike deterministic algorithms, cannot guarantee finding a global minimum. Because of this property of stochastic algorithms, further research has become popular in obtaining a better global minimum for different classifications of problems [3, 14]. Various sub-categories of stochastic algorithms have appeared, like probabilistic approaches [3], Monte Carlo approaches [3], evolutionary algorithms [3] and metaheuristics methods [5]. Each sub-category target different problems and shows various improvement on different type of functions. Some common well known algorithms include: particle swarm optimization (evolutionary algorithm) [11], differential evolution (metaheuristic method) [3], genetic algorithm (evolutionary algorithm) [3], cross-entropy method (Monte-Carlo algorithm) [3], and simulated annealing (probabilistic algorithm) [3]. Two algorithms used to solve the applications in Chapter 4 is Global Search and Particle Swarm Optimization.

### Global Search

*Global Search* (GS) is a hybrid heuristic algorithm that generates population points using the scatter search algorithm [7] and a local solver to optimize around the points. GS starts by locally optimizing around the initial point, $x0$, the user provides to the algorithm. If the local optimization converges various parameters are recorded

- initial point,
- convergent point,
- final object function value, and
- score value.

The *score value* is determined by taking the sum of the objection function value and any constraint violations. If the point is feasible then the score value is equal to the objective function value. Otherwise every constraint not satisfied adds on additional constraint violation value, typically a thousand. This is a form of a penalty function to avoid exploration around points that do not satisfy the constraints.

The algorithm will then generate trials points using the scatter search algorithm and evaluate each point for their score value. The point with the best score value is then optimized with the local solver. The same information is stored on this trial point as the initial point.

The algorithm then initialize the basins of attraction value and the radius. *Basins of attraction* are the heuristic assumption to be spherical. Thus two spheres are centred around the convergent points of the initial and best trial points with the radii being the distance between the start points to the convergent points of the local optimization. These estimated basins can overlap.

A local solver threshold is initialized to be less than the two convergent objective function values, if those points scare values are infeasible then the value is equal to score value of the first trial point.

Two counters are initialized to zero that are associated with number of consecutive trial points that lie with in a basin of attraction (counter per basin) and when a score value is greater than the local solver threshold.

The algorithm then continues to evaluate each trial point using the local optimizer if the following conditions hold

- condition 1

$$|x_i - b_j| > dr_j \tag{2.7}$$

  where $x_i$ is the $i$th trial point and $b_j$ is the $j$th basin of attraction centre; $d$ is the distance threshold factor, default value 0.75, and $r_j$ is the radius of $j$th basin of attraction.

- Condition 2

$$score(x_i) < l$$

  where l is local solver threshold.

- Condition 3 (optional) $x_i$ satisfies bound and inequality constraints.

If all conditions are met then the local solver runs on the trial point, $x_i$. If the local solver converges then the global optimum solution is updated if one of the following conditions is satisfied

$$\begin{gathered} |xc_k - xc_i| > T_x \max(1, |xc_i|) \\ or \\ |fc_k - fc_i| > T_f \max(1, |fc_i|) \end{gathered} \tag{2.8}$$

where $xc_k$ and $xc_i$ is every $kth$ convergent point and the convergent point for the $i$th trial point; $fc_k$ and $fc_i$ is every $k$th objective function value and objective function for the convergent point for the $i$th trial point; $T_x$ and $T_f$ are the $x$ tolerance and function tolerance, default $1\dot{1}0^{-}8$.

Likewise the basin radius and local solver threshold is updated if the local solver converges. The updates are as follows

- threshold is set to the score value at the trial point, and
- basin radius is set to the $xc_i$ to $x_i$ distance and maximum existing radius (if any).

If the local does not run on the trial point due to the conditions not being satisfied the two counters are incremented. The first counter is the basin counter, it increments for each basin $b_j$ that $x_i$ is in. The second counter is the threshold counter, it increments if $score(x_i) \geq l$ otherwise reset to 0. At the beginning of algorithm both counters are set to zero.

When each basin counter is equal to maximum counter value then basin radius is multiplied by one minus the basin radius factor and reset the basin counter to zero. If the threshold counter is equal to the maximum counter value then increase the local solver threshold to

$$l = l + p_f(1 + |l|) \tag{2.9}$$

where $p_f$ is a penalty threshold factor, then reset the counter to zero.

**Particle Swarm Optimization**

Particle Swarm Optimization (PSO) is a black box algorithm developed in 1995 by Kennedy and Eberhart [11]. This algorithm is a nature-inspired algorithm from a simplified social swarm model, where the algorithm mimics the social behaviour of flocking birds. Each particle is assigned a position, once evaluated a global best is assigned to a particle position then the other particles obtain a stochastic velocity to swarm to wards the global best particle. The stochastic velocity is based on experience of the specific particle and the knowledge of the swarm

$$v_{i,j}^{k+1} = v_{i,j}^{k} + c_1 r_1 (x_b^k - x_{i,j}^k) + c_2 r_2 (x_g^k - x_{i,j}^k) \tag{2.10}$$

where $x_{i,j}^k$ and $v_{i,j}^k$ are the $j$th component of the $i$th particle's position and velocity vector in the $k$th iteration; $r_1$ and $r_2$ are two random numbers, $x_b$ and $x_g$ are the best position the particle experienced and the global

best in the swarm; $c_1$ and $c_2$ are two parameters that represents the particle's confidence in itself and the swarm. The position of the particle is then updated

$$x_{i,j}^{k+1} = x_{i,j}^{k} + v_{i,j}^{k+1} \tag{2.11}$$

using the velocity obtained by Equation 2.10. To avoid premature convergence on local minima or over exploration of the particles Shi and Eberhart [15] introduced a new term known as the *inertia weight*, w. The inertia weight balances out the premature convergence and over exploration problems by influencing the previous velocity term

$$v_{i,j}^{k+1} = wv_{i,j}^{k} + c_1 r_1 (x_{local}^{k} - x_{i,j}^{k}) + c_2 r_2 (x_{global}^{k} - x_{i,j}^{k}). \tag{2.12}$$

This added term showed overall performance increase in the standard PSO algorithm. Then later Clerc [] used a constriction factor to insure convergence in the particle swarm. This altered the Equation 2.10 to

$$\begin{aligned} v_{i,j}^{k+1} &= \chi\left[v_{i,j}^{k} + c_1 r_1 (x_b^k - x_{i,j}^{k}) + c_2 r_2 (x_g^k - x_{i,j}^{k})\right] \\ \chi &= \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|} \; where \; \phi = c_1 + c_2, \; \phi > 4, \end{aligned} \tag{2.13}$$

this equation also prevents particle divergence. The schematic movement of the particle shown in Figure 2.2 follows the Equation 2.13.

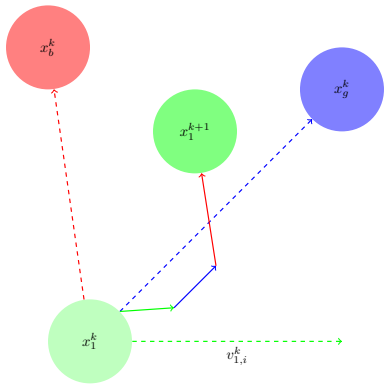



**Figure 2.2:** Schematic of the particles movement using Equation 2.12

When compared to the inertia weight Equation 2.10, Shi and Eberhart found they were equivalent in performance []. Thus either velocities align are used in the standard PSO algorithm:

**Algorithm 1** Particle Swarm Optimization

1: **for** each particle i **do**
2: **initialization** $x_i$, $v_i$, $xbest_i$ ▷ random value for $x_i$ and $v_i$ $xbest_i \leftarrow xbest_i$
3: **Evaluate** $f(x_i)$ ▷ evaluate the objective function at $x_i$
4: **Update** $xbest_i$ ▷ update if $f(x_i) < f(xbest_i)$
5: **end for**
6: **while** not termination condition **do**
7: **for** each particle i **do**
8: **update** xglobal ▷ update if $f(xglobal) < f(xbest_i)$
9: **calculate** $v_i$ ▷ Using one of the PSO velocity equations
10: $x_i = x_i + v_i$
11: **Evaluate** $f(x_i)$
12: **update** $xbest_i$
13: **end for**
14: **end while**

In the past recent years multiple variants of the PSO algorithm have been developed, collision-free PSO [], Discrete PSO [], Democratic PSO [], etc. One variant of PSO known as Global Convergence PSO, solve the crystal structure prediction problem discussed in Chapter 4. Van den Bergh and Engelbrecht [] developed the *Guaranteed Convergence PSO* (GCPSO) that particles preform a random search around the global best particle within a dynamic adapted radius. This encourages local convergence and addresses stagnation by randomly searching around the global best particle in an adaptive radius at each iteration. The GCPSO uses the Equations 2.12 and 2.13 to determine the particles velocity, $v_{i,j}^k$, and to update the inertia weight factor, $w$, over each iteration. Then updating the global best particles velocity by

$$v_{i_g}^{k+1} = -x_{i_g}^k + x_{i_g}^k + w^k v_{i_g}^k + \rho^k(1 - 2r_3^k) \tag{2.14}$$

where $i_g$ is the index of the particle that is most recently updated as the global best value. $r_3$ is a random number between $(0, 1)$. The search radius, $\rho^k$, is calculated by

$$\rho^{k+1} = \begin{cases} 2\rho^k, & \text{if} \tilde{s}^{k+1} > s_c, \\ \frac{1}{2}\rho^k, & \text{if} \tilde{a}^{k+1} > a_c \\ \rho^k, & \text{otherwise} \end{cases} \tag{2.15}$$

where $s_c$ is the success threshold and $a_c$ is the failure threshold. A success is indicated when Equation 2.14 results in an improved global best value, otherwise it is a failure. Each time a consecutive success occurs $s_c$ increase by one otherwise it is set back to zero if a failure occurs and vice versa for $a_c$.

# Chapter 3

# Software

## Computefarm

### Inspiration

As discussed in Chapter 2 global optimization solvers depend on the computational time, they also depend on fault tolerance of the objective function and resource consumption. In the event a black box function takes a long computational time, most global optimizations software offer parallel option. However, this is not beneficial when the function evaluation requires consumes a lot of resources to run the objective function. In this situation multiple function evaluations are ran simultaneously thus increasing the amount of resources need by a multiple of number of objective functions running at the same time. When this occurs, the demand for resources from the optimization becomes higher than the machine can provide and causes what is known as *resource contention*. This is where the motivation of computefarm comes from, a software application to parallelize global optimization solvers by distributing function evaluations to client computers. By distributing the evaluations out to client computers the resource contention is minimal and a number of search space points are evaluated simultaneously. Another feature of computefarm is that it handles failures in the client machines. If a client disconnects or fails to run the function evaluation, computefarm handles the failure by reassigning the evaluation to another client in hopes of success. This is more beneficial than on a single machine facing a failure that causes the whole optimization process to stop and to restart again. In some optimization cases this can be months of computational time lost due to a single failure. By using Computefarm for global optimization algorithms, the user is able to run the program in parallel, avoid high resource contention and obtain fault-tolerance in the function evaluations.

### Requirements

Computefarm is a distributed system to send out tasks to multiple client computers. In the case of global optimization a server distributes function evaluations to client computers. To keep overhead of using Computefarm to a minimal only three requirements are needed to run Computefarm, they are

- port number for socket connection (User provided),
- the runnable objective function script name (User provided), and

- list of population points to evaluate the objective function at on client machines (global optimization algorithm provided).

The first two points are passed into the solver and further passed to Computefarm for the initialization phase. The last point is passed in by the algorithm due to the metric it uses to determine which points in the domain of the objective function should be evaluated at each iteration step.

### Structure

The implementation of computefarm is based on the client-server model, such that the software farms unused or accessible client machines to wait upon tasks directed by the server machine. For global optimization this means the server delegates points in the search space to the client computers to evaluate a black box function. The clients then return the result to the server machine and wait on further positions to evaluate. The server then a provides a list of results to the global optimization solver from each evaluated point. In the case of a failure the server will receive a disconnect from the client machine then places the position back into the queue of positions needing to be evaluated. The server will later then provide the position to a client that is waiting on a new point to evaluate. This ensures fault tolerance in the computefarm software such that the global optimization solver does not need to restart if a client computer fails. An extra step is taken in Computefarm where client machines are reawakened to ensure the maximum number of client machines are available to be utilized by the server. Figure 3.1 shows the flow of a global optimization algorithm using Computefarm.

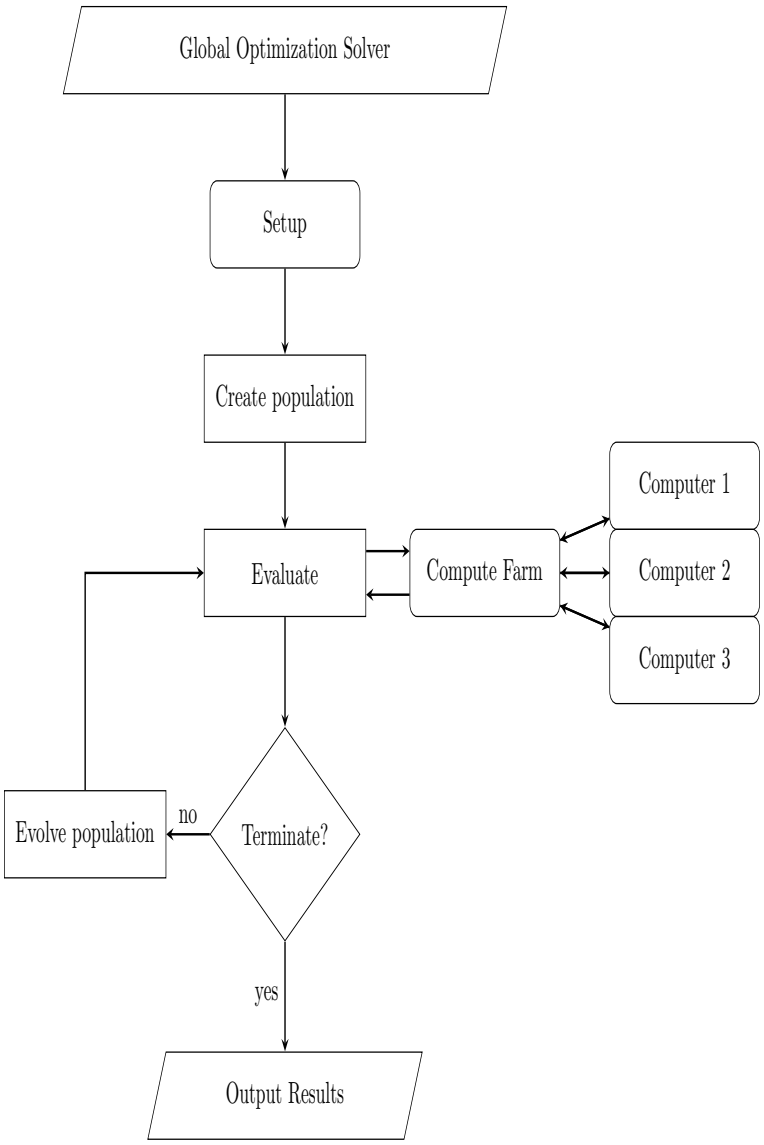



**Figure 3.1:** Process of a global optimization algorithm using Computefarm

Computefarm will take a list of population points and distributed them out to various client computers that will then return a result list to global optimization algorithm to further proceed in the optimization process. In this thesis Computefarm is applied to PSO to solve the applications in Chapter 4, by taking Alg. 1 an altered algorithm is implementd to use Computefarm Alg. 2.

**Algorithm 2** ComputeFarm Particle Swarm Optimization

**initialize** Computefarm ▷ initializes the setup of the software
**for** each particle i **do**
  **initialization** $x_i$, $v_i$, $xbest_i$ ▷ random value for $x_i$ and $v_i$ $xbest_i \leftarrow xbest_i$
**end for**
**Evaluate** Computefarm(X) ▷ evaluate the list of points X using Computefarm
**for** each particle i **do**
  **Update** $xbest_i$ ▷ update if $f(x_i) < f(xbest_i)$
**end for**
**while** not termination condition **do**
  **for** each particle i **do**
    **update** xglobal ▷ update if $f(xglobal) < f(xbest_i)$
    **calculate** $v_i$ ▷ Using one of the PSO velocity equations
    $x_i = x_i + v_i$
  **end for**
  **Evaluate** Computefarm(X)
  **for** each particle i **do**
    **update** $xbest_i$
  **end for**
**end while**

This algorithm is applied in the PSO variants in the software *pythOPT*, a problem solving environment, creating a distributed PSO version called Computefarm Particle Swarm Optimization (CFPSO).

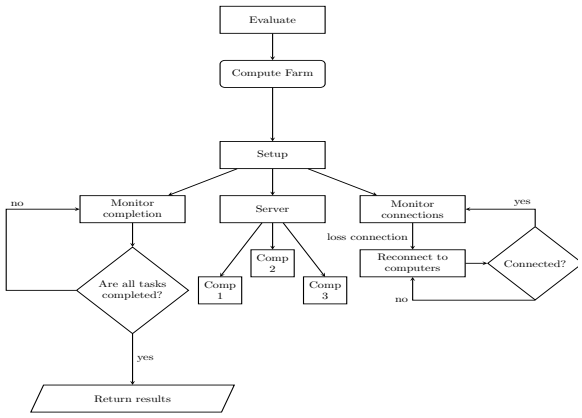
To further understand the implementation steps taken in Computefarm, Figure 3.2 shows the overall design of the distributed system.

During the setup phase of Computefarm, three POSSIX *threads* are used to run the functions

- Monitor completion,
- Server, and
- Monitor connections.

Threads are used in this software because of the shared memory properties they have that allows for a form of message passing in the program. Each thread function takes care of a single process that monitors a list to determine further actions. The lists that are shared in memory between the three threads are the medium for message passing in program.

Monitor completion function takes care of monitoring if all tasks are completed before returning the results to solver. The Server function creates a TCP socket and binds to any client computers communicating on the




1

**Figure 3.2:** Process of a global optimization algorithm using Computefarm

same port. TCP sockets were chosen because of their reliable connection handling. Threads are generated for each connected client to allow for concurrency in the server system. Each connection is handled by connection handler that will assign itself to a client machine by receiving the hostname of that machine. Once this is obtained the connection handler will change an client assigned flag for that machine from zero to one. After this it will continuously monitor particle list that contains a particle information structure. The particle information structure contains the following information

- particle number,
- particle position,
- function value,
- length of position,
- assigned flag, and
- completion flag.

While monitoring the particle list, if any particle is not assigned to a connection handler then it will change the assigned flag from zero to one indicating it has been assigned. To ensure multiple connection handlers do not assign themselves the same particle *mutexs* are used to ensure mutual exclusion. Once a connection

handler has assigned itself a particle, it will then send the particle position information to the client computer using the TCP socket. Because this portion of the program is written in programming language C, the particle position array length is also stored and sent to the client computer. The connection handler will then wait to receive a message from the client. This message will contain the resulting function evaluation. The connection handler will then store the function value for the specific particle and change the finished flag from zero to one. In the case of disconnect of the client, the connection handler will de-assign itself from the particle by changing the assigned flag back to zero, changed the client assigned flag back to zero and exit. The work flow of the connection handler is described in the following Figure 3.3.

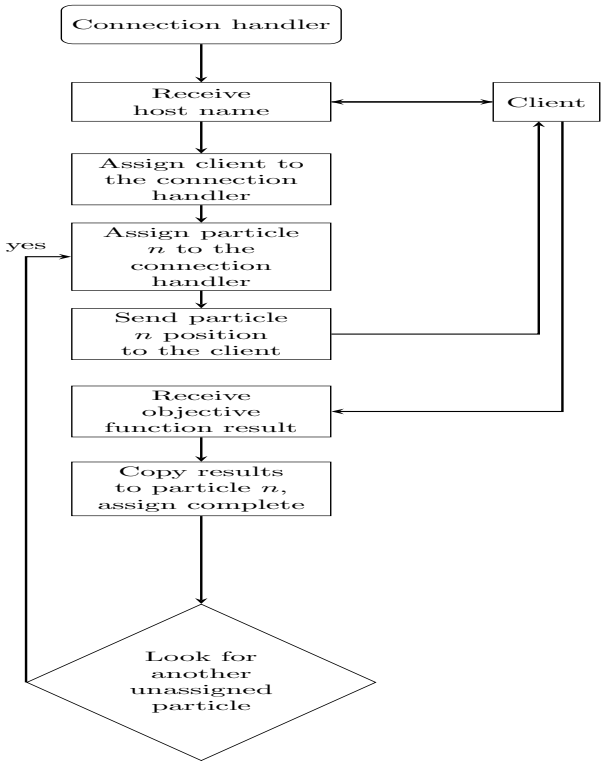



**Figure 3.3:** Computefarm connection handler work flow

The Monitor connection function awakens client machines to run the client script to connect to the server and monitor any machines that disconnect. When machines disconnect a separate thread function, reawaken clients, will attempt to awaken any known disconnected machines every $N$ minutes (default ten minutes). This ensure the maximum number of client machines connected to the server every $N$ minutes from start up. Disconnected machines are determined by assigned flag in client structure that contains the host name of the machine and assigned flag. When the assigned flag is zero the reawaken function will attempt to awaken that client machine.

In the following second phase of Computefarm the Server and Monitor connections threads will be kept alive while the Monitor completion will be called by the solver. The Monitor completion function will re-initialize the particle list with new positions and reset all other information on the particle, the following

process of the Server and Monitor connections will continously run to complete all function evaluations. Once every particle is evaluated then the Monitor completion will return the results back to the solver. This phase will be repeated multiple times until the final phase.

The final phase is when the solver is done its optimization process and is terminating. To ensure no leakage of memory or zombie threads, a termination function will be called by the solver to terminate all threads and running processes with in Computefarm.

# Optimization Database

## Inspiration

The Optimization Database is used to deal with various challenges present in the global optimization process. One challenge is monitoring the optimization process and if the problem is solved. Because global optimization solvers run until a termination condition is satisfied, a solution could be found sooner than later. Thus a Standard method of monitoring the optimization process is having the best value at each iteration step printed or saved in a log file. This can become unreliable if the file cannot save or printed values are lost to due to a failure. Also sorting through printed out information or files can be a lengthy process or needs extra code to do so. A way around this is using a database to store values based a policy. In the Optimization database there are three policies the user can chose from

- best value,
- every evaluation, or
- evert $n$th evaluation.

The database will store values based on the policy chosen at each evaluation and when inserted into the database it ensure the transaction is completed. If the transaction does fail it will report an error and try two more times to reconnect and insert the data. If this does not more then it will send a warning message to the user and continue the optimization process. Some situations when monitoring is need is when

- premature convergence occurs,
- performance testing, or
- obtained results.

Another challenge of global optimization is interest of obtaining $N$ best solutions, because global optimization is implemented to solve for the global minimum it is unusual to have a solver that returns the $N$ best global solutions. Therefore the Optimization database is used to sort through the data and obtain the $N$ best solutions that the solver explored. This becomes useful in the Chapter **??** for the crystal structure prediction

problem where there are metastable crystal structures. The metastable structures are of an interest because they can also have interesting properties. An example of this is diamond, it is a metastable structure to Carbon with the stable structure being graphite. In a global optimization, graphite is the global minimum because it has the lowest total energy. However, diamond being the second lowest energy has the property of being very hard and used in multiple research experiments to define hardness and compare hardness to other structures. Thus the Optimization Database is used to obtain the $N$ lowest energy crystal structures for various compounds. Another data storing problem of global optimization is obtain extra information on the objective function. The extra information can be

- information on the data, for example the symmetry group of a predicted crystal structure, or
- sorting information on various instances, for example the time duration it takes to correct for errors in a quantum component discussed in Chapter **??**.

The Optimization database is used to store information on the optimization of the objective function and be flexible to the user to change storage policies and store extra information. This allowing for easier, reliable data collection on the global optimization process.

### Requirements

The Optimization Database is used to store information for optimization problems, this includes local and global solvers. In this thesis the database is used only for global optimizations. The database software as primarily one user case, to store information about the global optimization. It is implemented to setup and create tables for users will no prior knowledge of databases and it can change what information is stored in the tables. The primary information needed is

- PostgreSQL database information,
- Problem name, and
- global optimization settings.

### Schema

The Optimization Database is implemented for users with no prior knowledge of databases. On use of the software it will create a local PostgreSQL database if the user does not have a local or remote database setup. It will then automatically setup two tables, settings and problem table. The settings table stores information about the settings used for the global optimization with the default columns

- global optimization method,
- lower bound,

- upper bound,
- seed, and
- note on simulation.

Columns that are included and maintained by the database is

- primary key id,
- status, and
- check in time.

The primary key is used to associate any instance of the problem that will be storing its data in the problem table. The status column is updated by the software to keep status if a simulation is still running. Whenever the database is updated or inserted into, the check in time will be automatically updated to the current time. This check in time can later be used to determine if the simulation has been running for the past $N$ hours. In combination with a automatic email component to the software the user can be notified on result and the status of the optimization daily.

The problem table is the record of data on the objective function that by default stores

- Foreign key id,
- $x$ position,
- $f$ (function evaluation result),
- evaluation number, and
- insertion time.

The user can also chose to have extra columns to store other data about the objective function. As mentioned before this can be used for quicker sorting methods between instances or properties of the simulation.

Figure 3.4 represents the database schema used by the software where one settings table has multiple instances of problem tables.


**Figure 3.4:** Optimization Database schema.

# Chapter 4

# Applications

# Chapter 5

# Conclusion

I conclude that I have solved the problem!

# References


[1] G. Accary, D. Morvan, and S. Me. The Human Line-1 Retrotranspsons Creates DNA Double Strand Breaks. *Fire Safety Journal*, 93(5):173–178, 2008.

[2] C. S. Adjiman. A global optimization method, aBB, for general twice-differentiable constrained nlps. *Journal of Chemical Information and Modeling*, 53(9):1689–1699, 2013.

[3] H. Aguiar and O. Junior. *Evolutionary Global Optimization , Manifolds and Applications*. Springer, 2016.

[4] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, a. N. Korotkov, a. N. Cleland, and J. M. Martinis. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

[5] U. Can and B. Alatas. Physics Based Metaheuristic Algorithms for Global Optimization. *American Journal of Information Science and Computer Engineering*, 1(3):94–106, 2015.

[6] J. Ghosh, A. Galiautdinov, Z. Zhou, A. N. Korotkov, J. M. Martinis, and M. R. Geller. High-fidelity controlled-$\sigma^Z$ gate for resonator-based superconducting quantum computers. *Physical Review A - Atomic, Molecular, and Optical Physics*, 87(2):1–19, 2013.

[7] F. Glover. A Template for Scatter Search and Path Relinking. *Artificial Evolution*, 1363(February 1998):2–51, 1998.

[8] Y. Hung and W. Wang. Accelerating parallel particle swarm optimization via GPU. *Optimization Methods and Software*, 27(1):33–51, 2012.

[9] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[10] A. Kaveh and BOOK. *Advances in Metaheuristic Algorithms for Optimal Design of Structures*. Springer, 2014.

[11] J. Kennedy and R. Eberhart. Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 4:1942–1948 vol.4, 1995.

[12] L. Liberti. Introduction to global optimization. *Ecole Polytechnique*, 2000.

[13] A. Mathematics. Global Convergence of a Class of Trust Region Algorithms for Optimization with Simple Bounds Author ( s ): A . R . Conn , N . I . M . Gould , Ph . L . Toint Published by : Society for Industrial and Applied Mathematics Stable URL : http://www.jstor.org/st. *Society*, 25(2):433–460, 2008.

[14] J. D. Pintér. Global Optimization: Software, Test Problems, and Applications. *Handbook of Global optimization*, 2:515–569, 2002.

[15] Y. Shi, R. Eberhart, Y. Shi, M. Clerc, A. Kaveh, A. Zolghadr, Y. Wang, B. Li, T. Weise, J. Wang, B. Yuan, Q. Tian, T. Krink, J. S. Vesterstrom, J. Riget, and F. Glover. A modified particle swarm optimizer. *Artificial Evolution*, 1363(February 1998):1951–1957, 1998.

[16] M. Veltman. COMPUTER PItYSICS COMMUNICATIONS 3, SUPPL. (1972) 75-78. NORTtI-IIOLLAND PUBI ISIIINC COMPANY ALGEBRAIC. *Computer Physics Communications 3*, pages 75–78, 1972.

[17] E. Zahedinejad, S. Schirmer, and B. C. Sanders. Evolutionary algorithms for hard quantum control. *Physical Review A - Atomic, Molecular, and Optical Physics*, 90(3):1–10, 2014.

# Appendix A

# Sample Appendix

Stuff for this appendix goes here.

# Appendix B

# Another Sample Appendix

Stuff for this appendix goes here.