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

Abstract

Mathematical models are a gateway into both theoretical and experimental understanding. However, sometimes these models need certain parameters to be established in order to obtain the optimal behaviour or value. This is done by using an optimization method that obtains certain parameters for optimal behaviour that may be a minimum (or maximum) result. Global optimization is a branch of optimization that takes a model and determines the global minimum for a given domain. However, global optimizations can become extremely challenging when the domain 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 find the solution. To address these challenges, two pieces of software were developed that aided the solver in optimizing a black box problem. The first piece of software developed is called Computefarm, a distributed system that parallelizes the iteration step of a solver by distributing function evaluations to idle computers. The second piece of software is an Optimization Database that is used to monitor the function and to store extra information on functions and results. It is also used to prevents data from being lost due to an optimization failure.

In this thesis, both Computefarm and the Optimization Database are 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 bit. 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 the system to measure the states can occur. Therefore, error correction circuits 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 stabilize for minimal error. This design is optimized with the use of the Optimization Database and Computefarm to obtain the properties needed to achieve a low error rate. The second application is crystal structure prediction, in which the total energy of crystal lattice is minimized to obtain its stability. Stable structures of carbon and silicon dioxide are obtained by using Computefarm and the Optimization Database. These software packages establish and store various stable structures and post-processing data for classification of the structure. The database is then used to obtain most stable structures of carbon and silicon dioxide for further research.

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

PSO	Particle Swarm Optimization
GCPSO	Guaranteed Convergence PSO

Chapter 1

Introduction

Optimization is a process to obtain a minimal (or maximal) result for a defined problem. Problems can range from simple functions like $f(x) = x^2$ to complex models (e.g., minimizing the water saturation in a fuel cell). By optimizing the objective functions, parameters for a given objective function are solved to minimize the result. This can lead to new discoveries in research and further the development of new technologies.

Optimization can be local or global. Local optimization searches in a given neighbourhood around a provided candidate solution. The search obtains a local minimum value in the neighbourhood of the domain based on satisfying convergence properties. When searching for a minimum in a case where the local minimum is not guaranteed to be the global minimum, a global solver is needed. Unlike the local solver, a global optimization searches the whole domain (rather than a particular neighbourhood). There are two types of global optimization: deterministic and stochastic [15]. These types of global optimizations have found global solutions for applications including chemical equilibrium, nuclear reactors, curve fitting, vehicle design and cost [18], and many others.

Deterministic algorithms guarantee a global minimum, but they might take a large amount of time to do so insofar as they search the whole domain. In the worst cases, these algorithms have complexities of exponential time. In cases where large amounts of time are not practical, stochastic algorithms are used instead.

Stochastic algorithms cannot guarantee a global minimum because of the adaptive random searches they deliver. However, they can determine a good candidate minimum in an amount of time specified by the user. Researchers 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 perform better on specific types of objective functions. Some examples of such algorithms are Particle Swarm Optimization [12], Genetic Algorithm [3], and Simulated Annealing [3].

Various types of objective functions lead to various challenges for the global solver. One kind of challenge is premature local convergence, which happens when a global solver gets stuck exploring a local minimum. This wastes time that could be spent finding a global minimum; it can also lead to longer optimization times for deterministic algorithms, or to a poor global candidate for stochastic algorithms. Objective functions that have the potential for premature local convergence are non-convex functions for which local minima are not guaranteed to be global minima. Thus, a solver gets stuck in a local minimum, making it difficult to find a global minimum value.

Failure can also occur in the objective function itself. The optimization process might face a graceful shutdown that requires restarting the process and re-evaluating the objective function in hopes a failure does not occur. One type of failure is the result of a resource contention. Resource contention can occur when both the parallel global optimization algorithms and the objective function use up various resources (processors and memory) or when multiple objective functions that demand more resource power are running simultaneously. Depending on the machine and the scheduling policy, resource contention can lead to serial optimization or failure.

Another challenge for global optimization involves monitoring the objective function to determine whether further action is needed, or in order to obtain extra information. The user has the flexibility to change various properties of the function. This can affect the sensitivity of the optimization process, often increasing or decreasing the time required to obtain a global minimum. The user may also want to derive external information from the objective function, such as

- specific properties of the model,
- post processing of data, or
- the top N best results the algorithm has found.

The contributions to solving these challenges are the software Computefarm and the Optimization Database. Computefarm is a distributed system that uses idle computer resources on various client computers to run multiple function evaluations at once. It can speed up the iteration process of the given solver, and thereby speed up the search for the global minimum in a problem. It is also fault tolerant to failures of a farmed computer by allowing the global optimization process to continue without the need to restart.

The Optimization Database is a flexible database that used by the model to store the results and any extra information pertaining to the simulation or post processing of the data is stored into the database. 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 the 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 the testing of desired reliability quite costly. In light of this, several models have been designed to simulate a quantum error correction circuit for the purpose of determining the effect of error correction on a given n-qubit system. To ensure high reliability (also known as fidelity) of the circuit design, the circuit parameters are optimized so that the model returns the desired fidelity of 99.99% [4, 7]. This is the highest modelled fidelity needed for a circuit design to achieve the in experimental fidelity of 99.9% that cannot improve any higher due to external noise that simulation cannot account for. In this thesis the 3-qubit and 4-qubit systems are solved using the global optimization and the software applications.

Crystal structure prediction has been used in the past decade to approximate the most stable structures 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 temperature and pressure can be taxing. Researchers thus use Ab Initio structure codes to calculate properties of the given lattice in the hope 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. In this thesis, various crystal structures (Carbon, Silicon, and Silicon dioxide) are found by optimizing for the minimal energy and storing the top stable structures in the database.

The remainder of this thesis is structured as follows. Chapter 2 gives the background on global optimization algorithms used in this thesis. 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, as well as how the software was used to aid in solving these problems. Final Chapter 5 summarizes the results obtained for the two applications and the benefits the software prioritized for solving the applications.

Chapter 2

Global Optimization

Global optimization is a widely-used method in engineering, chemistry, and economics, among other fields, to obtain the extreme minimum (or maximum) value of a function. We can represent real-world problems in the following form:

$$x^* = \arg\min_{x \epsilon D} f(x) f : D \to \mathbb{R} \tag{2.1}$$

$$D = x, x\epsilon\mathbb{R}^N, l_n \leq x_n \leq u_n, n = 1, \ldots, N \tag{2.2}$$

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 nth 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 (2.2) represents an *unconstrained global optimization* expression; to add constraints the form would include

$$\begin{aligned} &\textit{subject to} \\ &g(x) \leq 0, \\ &h(x) = 0 \end{aligned} \tag{2.3}$$

This is known as *constraint global optimization.*

In the 1950s, exhaustive searches like the Simplex algorithm [15] used this form (2.2). The global minimum was obtained by searching each point in the domain. This method of searching did guarantee the global minimum given enough time to search every possible combination. However, as objective functions became more complex and domains became larger, exhaustive searches became impractical. Researchers have recently begun exploring characterizations of the objective function, looking at constraints and algorithms to develop more efficient methods of obtaining 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

$$\lim_{x \to b} f(x) = f(b) \tag{2.4}$$

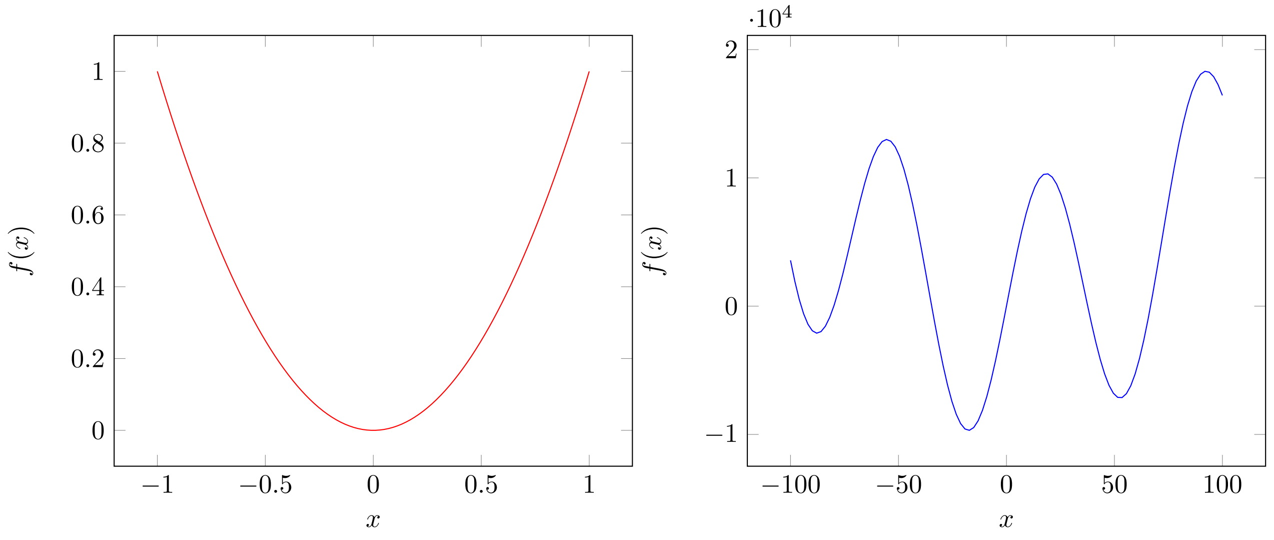
where x and b are independent points of each other. If (2.4) is satisfied for every point in the set b then the function is *continuous*, otherwise it is *discontinuous*. If the set is distinct and unconnected then the function is *discrete*, typically involving integers or whole numbers. In global optimization, 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 [11] and Branch and Bound algorithm [15], in which numbers are rounded up 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 to a minimum. In the case of non-smooth functions it becomes difficult to converge to a global minimum because of inconsistency of the landscape. One example of a non-smooth function is a probabilistic function that cannot 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, such as hybridizing particle swarm optimization [11]. *Convexity* is defined as the curvature of the function. A function is *convex* if for any $x_1, x_2 \epsilon X$ that follows

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2),\ 0 \leq \lambda \leq 1. \tag{2.5}$$

If the (2.5) does not hold, the function is *non-convex*. When a function is convex, all local minima are global minima, and 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, and the whole domain needs to be searched in order to guarantee that one of local minima is a global minimum. This is shown in Figure 2.1.



(a) Convex function.

(b) Non-convex function.

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

$$\bigtriangledown f(x) = 0, \tag{2.6}$$

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 the Multi-Level Single Linkage algorithm [15].

computational time is the time it takes to evaluate the objective function at a given point. In global optimization, this is a factor for optimization time and depends on whether concurrency is needed. To speed up the optimization time the objective function can be evaluated in parallel, if possible, or with the use of a parallel algorithm, such as parallel particle swarm optimization [9].

Objective functions are the core of global optimization, and can aid in the optimization process or make it challenging. As discussed in this section, certain characteristics of objective functions can help us determine which algorithm to use. For situations in which an objective function's characteristics are unknown or cannot be provided, a *black box global optimization* solver is used. The term "black box" refers to a unknown objective function that takes in input information and returns a value. These are commonly known as *derivative-free* algorithms, including Genetic Algorithm [3], Differential Evolution [3], Particle Swarm Optimization [12], etc.

Constraints

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

$$g_i(x) = 0, h_j(x) \leq 0 \tag{2.7}$$

where ith and jth are constraints for global optimization algorithms. In the event that 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. 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. One example is the *augmented Lagrangian method* [17], in which explicit constraints are combined with the objective function to create a sub-problem that is optimized by a local solver. When the constraints are not met, the sub-problem will penalize the value and the process will be repeated until a solution is obtained.

Some algorithms place constraints internally on the objective function to aid in solving the problem. An example of this is the α-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 functions changes the function to convex. In this situation the bounds were changed until the function is convex to then converge to 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. Examples include:

- 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 has to be satisfied before termination of the global optimization process.

Algorithms

Global optimization algorithms have traditionally been divided into two main types, stochastic and deterministic.

Deterministic

Deterministic algorithms guarantee finding the global minimum by searching the whole domain with tight convergence properties [18]. In 1969, the Branch and Bound algorithm [15] was one of the most well-known algorithms for solving complex problems like the travelling salesman problem [15]. Other deterministic algorithms include interval optimization [1], algebraic techniques [21], and DIRECT [10]. One weakness of deterministic algorithms is they can take large amount of time to find the global minimum, which is not practical to the user.

Stochastic

Stochastic algorithms, developed in the seventies, use adaptive random methods to obtain a feasibly close global minimum solution in a reasonable amount of time to the user. Unlike deterministic algorithms, stochastic algorithms cannot guarantee finding a global minimum. Because of this property of stochastic algorithms, further research has been directed towards obtaining better global minima for different classifications of problems [3, 18]. Various sub-categories of stochastic algorithms have appeared, including probabilistic approaches [3], Monte Carlo approaches [3], evolutionary algorithms [3] and metaheuristic methods [5]. Each sub-category targets a different problem and shows various improvements on different types of functions. Some well-known algorithms include: particle swarm optimization (evolutionary algorithm) [12], 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 are 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 [8] and a local solver to optimize around the points. GS starts by locally optimizing around the initial point, $x0$, which the user provides to the algorithm. If the local optimization converges, various parameters are recorded, including:

- 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 an additional constraint violation value, typically one thousand. This is a form of a penalty function, the purpose of which is to deter exploration around points that do not satisfy the constraints.

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

The algorithm then initializes the center points and radii of the basins of attraction. We make the heuristic assumption that *basins of attraction* are spherical. Two spheres are thus centred around the convergent points of the initial and best trial points, with the radii being the distance from 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' score values are infeasible then the value is equal to the score value of the first trial point.

Two counters (one counter per basin) are initialized to zero. These counters are associated with the number of consecutive trial points that lie within the respective basins of attraction, and record the number of times a score value is greater than the local solver threshold.

The algorithm then proceeds to evaluate each trial point using the local optimizer, provided the following conditions hold:

- Condition 1

$$|x_i - b_j| > dr_j \tag{2.8}$$

where x_i is the ith trial point, b_j is the jth basin of attraction center, d is the distance threshold factor (with a default value of 0.75), and r_j is the radius of the jth basin of attraction.

- Condition 2

$$score(x_i) < l$$

where l is the 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, provided one of the following conditions is satisfied:

$$\begin{gathered} |xc_k - xc_i| > T_x \max(1, |xc_i|) \\ or \\ |fc_k - fc_i| > T_f \max(1, |fc_i|) \end{gathered} \tag{2.9}$$

where xc_k and xc_i are the kth and ith convergent points for the kth and ith trial points, respectively; fc_k and fc_i are the objective function values for the kth and ith convergent points, respectively; ; T_x and T_f are the x tolerance and function tolerance, respectively, their default values being $1\dot{1}0^-8$.

The basin radius and local solver threshold are likewise 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 lesser of (i) the distance from xc_i to x_i, and (ii) the maximum existing radius (if any).

If the local solver 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, which increments for each basin b_j that x_i is in. The second counter is the threshold counter, which increments if $score(x_i) \geq l$ and otherwise resets to 0. At the beginning of the algorithm both counters are set to zero.

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

$$l = l + p_f(1 + |l|) \tag{2.10}$$

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

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a black box algorithm developed in 1995 by Kennedy and Eberhart [12]. This algorithm was inspired by 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; the other particles swarm towards it at a stochastic velocity based on the experience of the specific particle and the knowledge of the swarm, as follows:

$$v_{i,j}^{k+1} = v_{i,j}^{k} + c_1 r_1 (x_b^k - x_{i,j}^k) + c_2 r_2 (x_g^k - x_{i,j}^k) \tag{2.11}$$

where $x_{i,j}^k$ and $v_{i,j}^k$ are the jth component of the ith particle's position and velocity vector in the kth 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, respectively; and c_1 and c_2 are two parameters that represents the particle's confidence in itself and the swarm, respectively. The position of the particle is then updated

$$x_{i,j}^{k+1} = x_{i,j}^{k} + v_{i,j}^{k+1} \tag{2.12}$$

using the velocity obtained by (2.11). To avoid premature convergence on local minima or over-exploration of the particles, Shi and Eberhart [**?**] introduced a new term known as *inertia weight*, w. The inertia weight balances out the premature convergence and over-exploration problems by influencing the previous velocity term

$$v_{i,j}^{k+1} = w v_{i,j}^{k} + c_1 r_1 (x_{local}^k - x_{i,j}^k) + c_2 r_2 (x_{global}^k - x_{i,j}^k). \tag{2.13}$$

This added term showed overall performance increase in the standard PSO algorithm. Later, Clerc [16] used a constriction factor to ensure convergence in the particle swarm. This altered the (2.11) to

$$\begin{gathered} v_{i,j}^{k+1} = \chi\big[v_{i,j}^{k} + c_1 r_1 (x_b^k - x_{i,j}^k) + c_2 r_2 (x_g^k - x_{i,j}^k)\big] \\ \chi = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|} \ \textit{where}\ \phi = c_1 + c_2,\ \phi > 4, \end{gathered} \tag{2.14}$$

This equation also prevents particle divergence. The schematic movement of the particle shown in Figure 2.2 follows the (2.14).

When compared to the inertia weight (2.11), Shi and Eberhart found theirs was equivalent in performance [19]. Thus either velocities are used in the standard PSO algorithm:

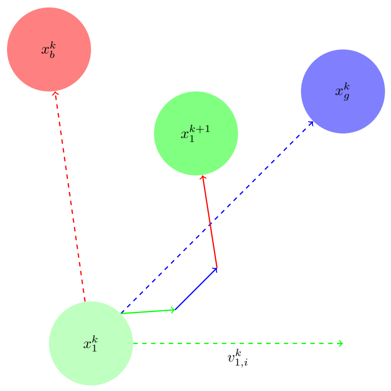


Figure 2.2: Schematic of the particle movement using (2.13)

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
```

Over the past few years multiple variants of the PSO algorithm have been developed: Collision-free PSO [14], Discrete PSO [13], Democratic PSO [11], etc. One variant of PSO, known as Global Convergence PSO, solves the crystal structure prediction problem discussed in Chapter 4. Van den Bergh and Engelbrecht [20] developed the *Guaranteed Convergence PSO* (GCPSO), whereby particles perform 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 (2.13) and (2.14) to determine the particle's velocity, $v_{i,j}^k$, and to update the inertia weight factor, w, over each iteration. The global best particle's velocity is then updated by

$$v_{i_g}^{k+1} = -x_{i_g}^k + x_{i_g}^k + w^k v_{i_g}^k + \rho^k(1 - 2r_3^k) \tag{2.15}$$

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

$$\rho^{k+1} = \begin{cases} 2\rho^k, & \text{if} \check{s}^{k+1} > s_c, \\ \frac{1}{2}\rho^k, & \text{if} \check{a}^{k+1} > a_c \\ \rho^k, & \text{otherwise} \end{cases} \quad (2.16)$$

where s_c is the success threshold, and a_c is the failure threshold. A success is indicated when (2.15) results in an improved global best value; otherwise it is a failure. Each time a consecutive success occurs, s_c increases by one; otherwise it is reset to zero (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.

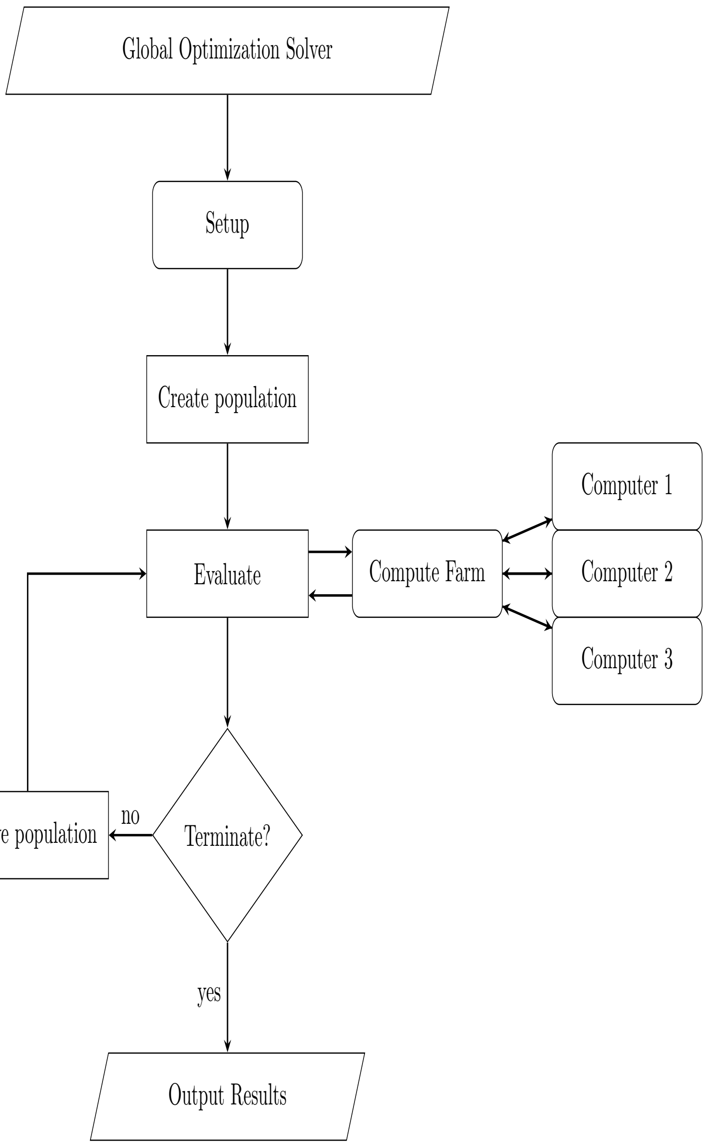


Figure 3.1: Process of a global optimization algorithm using Computefarm

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
 2: for each particle i do
        initialization $x_i$, $v_i$, $xbest_i$     ▷ random value for $x_i$ and $v_i$ $xbest_i \leftarrow xbest_i$
 4: end for
    Evaluate Computefarm(X)                        ▷ evaluate the list of points X using Computefarm
 6: for each particle i do
        Update $xbest_i$                           ▷ update if $f(x_i) < f(xbest_i)$
 8: end for
    while not termination condition do
10:     for each particle i do
            update xglobal                         ▷ update if $f(xglobal) < f(xbest_i)$
12:         calculate $v_i$                        ▷ Using one of the PSO velocity equations
            $x_i = x_i + v_i$
14:     end for
        Evaluate Computefarm(X)
16:     for each particle i do
            update $xbest_i$
18:     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).

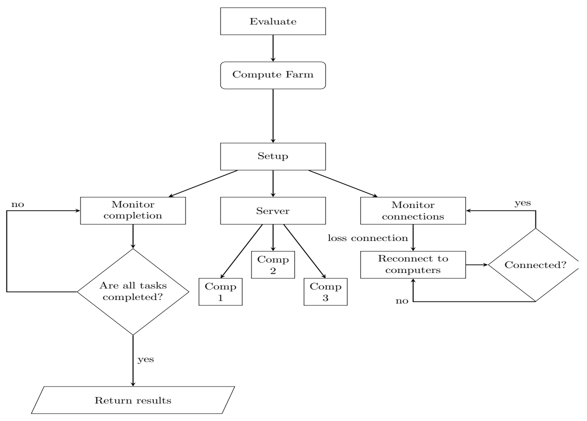
To further understand the implementation steps taken in Computefarm, Figure 3.2 shows the overall design of the distributed system.

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.

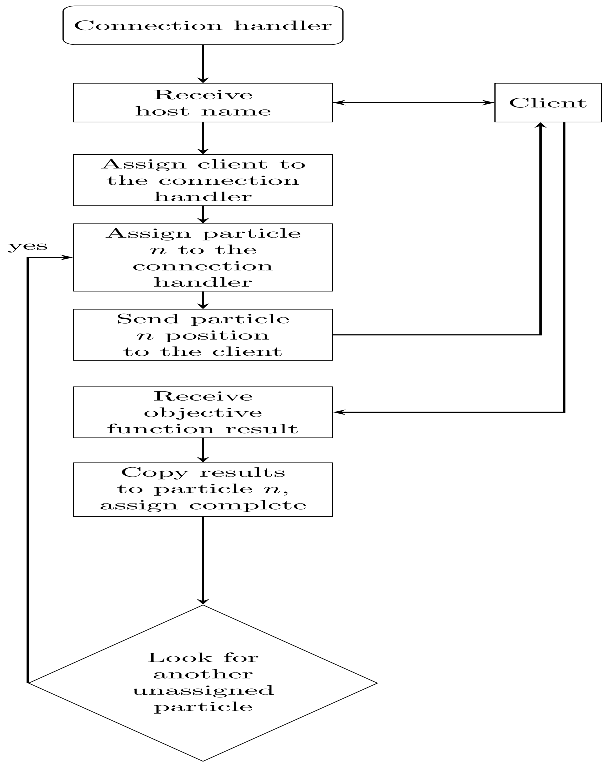

Connection handler
Receive host name
Client
Assign client to the connection handler
yes
Assign particle n to the connection handler
Send particle n position to the client
Receive objective function result
Copy results to particle n, assign complete
Look for another unassigned particle

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 nth 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 4.1 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.

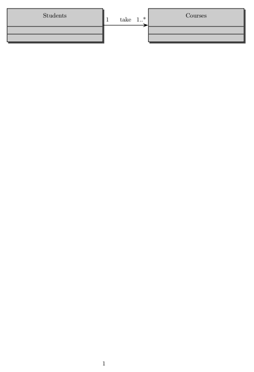


Figure 3.4: Optimization Database schema.

Chapter 4
Applications

Quantum computers are promising technology that is currently being used today by D-Waves, Nasa, google and IBM. Multiple institutes also focus on researching quantum computers and their exponential ability to represent information. This is obtained from the representation of a quantum bit also known as a *qubit*, a qubit is subatomic particle that is measured based on a known property like spin or an energy state. These properties represent states zero or one, same as a classical bit. However, qubits also have *superposition* or *entangled* states that the known states become linear combinations. For example in a classical computer two bits can represent one of the four possible values

$$00, \tag{4.1}$$

$$10, \tag{4.2}$$

$$01, \tag{4.3}$$

$$11. \tag{4.4}$$

To make up these combinations it takes two pieces of information, the first bit value and the second, thus a classical computer takes N bits of information. In a quantum computer two qubits can represent one of the same possible classical computer values. However, qubits are non-deterministic meaning they can be in any combination of states at any point in time known as *superposition.* Therefore to encode the qubits into a specific states, probabilities are given to the qubit system

$$\alpha|00\rangle, \tag{4.5}$$

$$\beta|10\rangle, \tag{4.6}$$

$$\gamma|01\rangle, \tag{4.7}$$

$$\delta|11\rangle. \tag{4.8}$$

$$\tag{4.9}$$

Thus four pieces of information are given to system giving quantum computers the exponential advantage of 2^N (N qubits) bits of information. Because of this advantage, a combination of calculation steps are done simultaneously or reduced to less steps with the use of more information. A strong example of this is prime factorization, where it is NP-hard problem for classical computers. In quantum computers the number fifteen has been solved using the Shor's algorithm on a three qubit system in seconds by Lucero [**?**]. With Dattani and Bryans [6] factorizing the number 56153 using a four qubit system. However, as it is mentioned in both papers Lucero [**?**] and Dattani [6] the error of the qubit systems prevents further work into factorization of higher numbers. With high error rates the success rate becomes lower. This becomes a difficult challenge for quantum cryptography that utilizes the fact quantum computers can factor high prime numbers in an efficient amount of time. However, cryptography needs fault tolerance to ensure the security of encryption and the ability to decrypt. To insure fault tolerance error correction components have been design to correct for any error in the qubit system. The two main errors in a qubit system is decoherence and measurement error. *Decoherence* is the loss of energy in the qubit system caused by interaction background particles. When two particles interact unintentionally this causes a loss of energy in the system, this energy is the information passed into the system, when the energy is lost the information is too. The other error that can occur is measurement error, this is caused by noise in the system. To correct for these errors pulses of energy are used to stabilize the qubit system because cloning qubit states is not possible due to their non-deterministic behaviour. However, stabilizing the system is constricted to specific amount of time because of decoherence. When enough outside interactions happen the qubit system can no longer be stabilized as their is too much loss of information. Another challenge is decoherence is not easy to model as it is not easy to predict what background particles will interact with the qubits and how the interaction will affect the system. Noise error is easier to model as it is the noise in qubit system and in logic gate utilizing the system. Therefore by optimizing the error correction circuit design to have a qubit system simulated to have noise error be fault tolerant, then experimentally that circuit can be further optimized for decoherence.

In this section the Controlled Controlled Controlled Not (CCCNOT) gate is optimized to obtain a feasible error correction percentage known as *intrinsic fidelity*. A simulation of the CCCNOT gate is optimized to achieve a feasible intrinsic fidelity of 99.99% to guarantee the highest on average experimental gate fidelity of 99.9% [4]. The CCCNOT simulates a four superconducting charge qubits known as *transmons*. Transmons are used because of the reduced sensitivity to charge noise to aid in reducing error. The states zero and one are represented as energy states of the transmon, j. Each transmon location in the system is represented as k that receive pulses from the error correction circuit over a given amount of time, t. The shift in frequencies sent to each transmon is represented as $\Delta_k(t)$ (bounded between -2.5 and 2.5 MHz) and anharmonicity of the shift in frequency of each transmon is represented as η_{jk} that is measured to be 200 MHz for the circuit. The energy of each k^{th} transmon at each energy level j is

$$E_{kj} = h(j\Delta_k(t) - \eta_{jk}), \tag{4.10}$$

where h is planks constant. As mentioned earlier qubits can entangle with one another, this interaction is represented as a nearest-neighbour coupling strength, g_k, between each kth and $(k+1)th$ transmon. The coupling strength is set as 30 MHz in simulation.

The energy transition between transmons states is then represented as the n transmon system (in the case of a four qubit system $n = 4$)

$$\frac{\hat{H}\big(\delta_k(t)\big)}{h} = \sum_{k=1}^{n}\sum_{j=0}^{n} E_{kj}|j\rangle_k\langle j|_k + \sum_{k=1}^{n-1}\frac{g_k}{2}(X_kX_{k+1} + Y_kY_{k+1}), \tag{4.11}$$

where X and Y are the coupling operators [7]

$$X_k = \sum_{j=1}^{n}\sqrt{j}|j-1\rangle_k\langle j|_k + hc, \tag{4.12}$$

$$Y_k = -\sum_{j=1}^{n}\sqrt{-j}|j-1\rangle_k\langle j|_k + hc, \tag{4.13}$$

and hc is the Hermitian conjugate. The couple operators (4.12) represent the generalize Pauli spin matrices for each kth transmon.

By knowing the Hamiltonian (4.11) of the transmons system the evolution operator of the system over a time t is represented as

$$U\big(\Delta_k(\Theta)\big) = Te^{\left\{-i\int_0^{\Theta} \hat{H}\big(\Delta_k(t)\big)dt\right\}}, \tag{4.14}$$

Where Θ is time duration of the error correction and T is the time ordered evolution operator [].

Because transmons states have to fall between zero or one (bias states) when observed, high energy levels are not considered when correcting the error. To do this a projection is taken on the evolution operator (4.14) to obtain the computation subspace

$$U_{\mathscr{P}}\big(\Delta_k(\Theta)\big) = \mathscr{P}U\big(\Delta_k(\Theta)\big)\mathscr{P}. \tag{4.15}$$

The computational subspace (4.15) is the performance metric intrinsic fidelity. To simplify this down to a single percentage value, the percentage of the intrinsic fidelity is represented as

$$\mathscr{F}\big(\Delta_k(\Theta)\big) = \frac{1}{N}\left|\mathrm{Tr}\Big(CCCNot^{\dagger}U_{\mathscr{P}}\big(\Delta_k(\Theta)\big)\Big)\right|, \tag{4.16}$$

where CCCNot is the ideal gate [].

This model represent the objective function to optimize the frequency shifts, $\Delta_k(t)$, for a n-transmons system to obtain feasible intrinsic fidelity of 99.99% for a minimal duration time Θ.

The optimization problem is presented as follows

$$0.9999 \leq f(x), \tag{4.17}$$

where a feasible solution is just needed.

To solve this problem for the four-qubit ($n = 4$) and three-qubit ($n = 3$) case we used *MATLABs* global search algorithm from the global optimization toolbox with a non-linear constraint to represent the feasibility condition (4.17). To minimize the duration time of the simulation a brute force method is used by simply solving each case with a duration time and lowering it to find the minimal value. The Optimization Database is used in the optimization process to monitor multiple duration time cases and the progress of global optimization. By using this method to solve the problem the following results for the four-qubit case are shown in Table **??** and the three qubit case result shown in Table **??**.

The feasible pulse for minimal duration time of 65 nanoseconds for the four qubit case is shown in Figure **??**.

The feasible pulse for minimal duration time of 23 nanoseconds for the three qubit case is shown in Figure **??**.

By obtaining these results, error correction circuits using the CCCNot gate are developed for further experimental optimization to account for decoherence error to be used in quantum computers.

Crystal structure prediction

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] N. S. Dattani, N. Bryans, E. Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariantoni, A. Megrant, P. O. Malley, D. Sank, A. Vainsencher, J. Wenner, T. White, Y. Yin, A. N. Cleland, and J. M. Martinis. Quantum factorization of 56153 with only 4 qubits. *Nature Physics*, 8(143):1–6, 2014.

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

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

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

[10] 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.

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

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

[13] J. Kennedy and R. Eberhart. A discrete binary version of the particle swarm algorithm. *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, 5:4104–4108, 1997.

[14] T. Krink, J. S. Vesterstrom, and J. Riget. Particle swarm optimisation with spatial particle extension. *Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002*, 2(February 1998):1474–1479, 2002.

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

[16] C. M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. *1999 ICEC, Washington DC*, pages 1951–1957, 1999.

[17] 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.

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

[19] Y. Shi and R. Eberhart. A Modified Particle Swarm Optimizer. *IEEE*, pages 69–73, 1998.

[20] A. Van den Bergh F, Engelbrecht. A new locally convergent particle swarm optimizer. *IEEE conference on systems, man and cybernetics*, 2002.

[21] M. Veltman. COMPUTER PItYSICS COMMUNICATIONS 3, SUPPL. (1972) 75-78. NORTtI-IIOLLAND PUBI ISIIINC COMPANY ALGEBRAIC. *COMPUTER PHYSICS COMMUNICATION 3*, pages 75–78, 1972.

Appendix A

Sample Appendix

Stuff for this appendix goes here.

Appendix B

Another Sample Appendix

Stuff for this appendix goes here.