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 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. However, 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 function and to store extra information on function and results.

In this thesis, both Computefarm and the Optimization Database were 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 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 stabilize to obtain minimal 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 to obtain and store various stable structures and post processing data for classification of the structure. The use of the database then is used to obtain the top N stable 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 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. Thus 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 [5]. These types of global optimizations have found global solutions for the applications chemical equilibrium, nuclear reactors, curve fitting, vehicle design and cost [6], and many others.

Deterministic algorithms guaranteed global a minimum however, 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 are then used.

Stochastic algorithms cannot guarantee a global minimum because of the adaptive random searches they do. However, they can determine a good candidate minimum in a specified amount of time from the user. Researches tend to use these algorithms if the exact global minimum is not needed to solve for 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 [**?**], Genetic Algorithm [], Simulated Annealing [].

Various type 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 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 a parallel global optimization algorithms uses various resources (processors and memory) and the objective function also uses up resources. When multiple objective functions are running simultaneously that demand more resource power, contention 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 needed to 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 user may want to know

- specific properties of the model,
- post processing of 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 the 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 [2, 4]. This is the highest modelled feasibility needed due to external noise when manufactured. This feasibility value has been obtained for the 3-qubit system [7]. 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 used in engineering, chemistry, economics and etc, for obtain the extreme minimum (or maximum) value. By formulating real world problems to obtain the global minimum in the form of

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

$$f : D \to \mathbb{R} \tag{2.2}$$

$$D = x, x\epsilon\mathbb{R}^N, l_n \leq x_n \leq u_n, n = 1, \ldots, N \tag{2.3}$$

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 2.1 represents an unconstrained global optimization expression, such that to add constraints the form would include

$$g(x) \leq 0, \tag{2.4}$$

$$g(x) = 0 \tag{2.5}$$

In the 1950's exhaustive searches like the Simplex algorithm [5] 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 obtain more efficient methods that can or obtain close to 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 can presents multiple characteristics that can determine the difficulty to optimize in the given domain, some examples are

- continuity,

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

Continuity is defined as

$$\lim_{x \to a} f(x) = f(a) \tag{2.6}$$

where x and a are independent points of each other. If 2.6 is satisfied for every point in the set a 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 can determine if a specific type of algorithm is needed. For discrete functions mixed integer algorithms are used as they pick points in integer domain. In the case of discontinuous functions, the objective function, space or constraints are modified to make the function either continuous or seem continuous to the solver. Examples of these modifications is

- generate a bad result value for gap or asymptotic regions,
- constrain the space to contain the continuous regions of the function,
- add constraints to avoid gap or asymptotic regions.

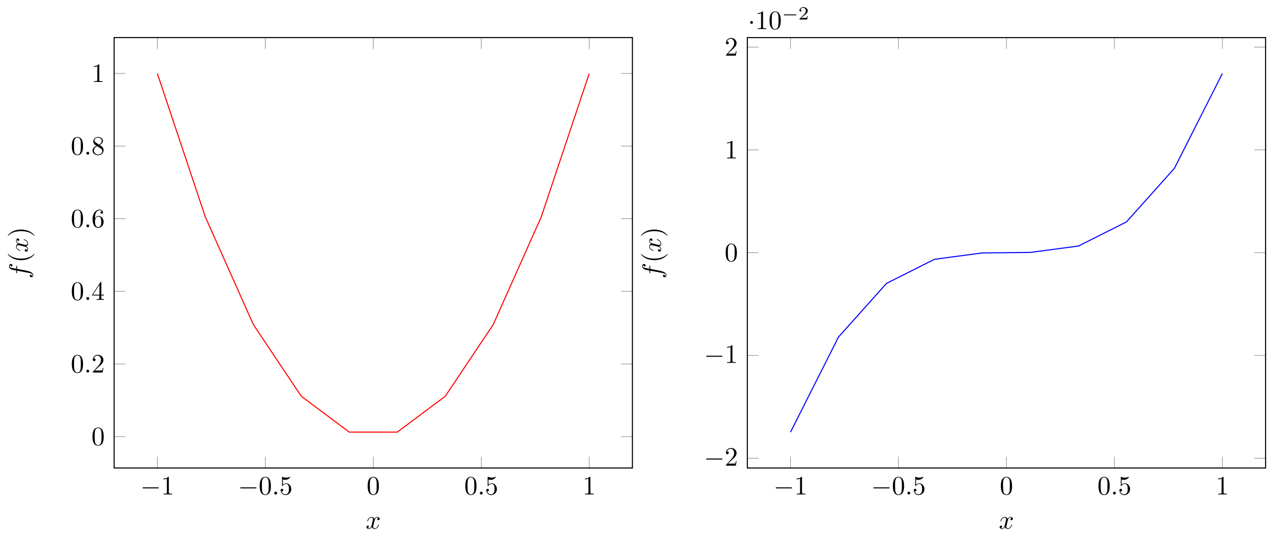
Smoothness is defined as the order of continuous derivatives the function can achieve. The higher the order of the continuous derivative the smoother the function is. For global optimization the smoother the function the easier it is to converge on a minimum. However, in the case of non-smooth functions for example objective functions based on probabilities, the data becomes noise. In this case the objective function or the algorithm need to smooth out the data by either averaging the results with in the objective function or the algorithm re-evaluates the point several times to obtain the best or averaged value. ¡add example of algorithm that does this¿

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

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2), \ 0 \leq \lambda \leq 1. \tag{2.7}$$

If the condition 2.7 does the hold the function is *non-convex*. The convexity of the function also indicates the difficulty to optimize the function, shown in Figure 2.1, the local minimum of a convex function is also the global minimum (or concave for local maximum), thus a local optimization solver can solves these types of functions

When the function is non-convex the local minimum is not guarantee to be the global minimum. In this case a global optimization solver is needed to search the whole domain to obtain the global minimum.



(a) **(b)**

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

Differentiable is defined if a functions derivative exists. If a functions derivative does exist, then it can be used to aid global optimization solver to converge to solution faster. The solver uses the derivative to determine the slope at a given point, if

$$f'(x) = 0, \tag{2.8}$$

then a local minimum or maximum is obtained.

In an objective function this is implemented by returning an extra output or implementing another function that returns the derivative to the solver.

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 come parallel if possible or the use of a parallel algorithm, for example parallel particle swarm optimization.

Objective functions are the core of global optimizations that can aid in the optimization process and make it challenging. As discussed in this section certain characteristics are also used to determine the algorithm needed to be used on the function. In the situation when then objective functions characteristics are unknown or cannot provide extra information like the derivative, 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.

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 condition, typically:

$$g_i(x) = 0, h_j(x) \leq 0 \tag{2.9}$$

where ith and jth condition value. In the event the constraints are not satisfied the algorithm will apply a penalty to 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 is occurs when algorithms does not provide the option to set explicit constraint functions. In this case the user will then implement a condition in the objective function that will return a penalized value. An algorithm that incorporates this is the *augmented Lagrangian method* developed by [], 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. To obtain global minimum global this algorithm, locally optimizes several regions of the search space to obtain the global minimum.

For algorithm conditions, these are set with in the algorithm to restrict the search space or internal algorithm parameters. An example of algorithm constraints is used in the α Branch and Bound algorithm developed by [], where internal constraints were placed on the lower and upper bounds of the objective function to segment sections that made the function convex. This is a form of *convex relaxation*, where certain manipulations to non-convex functions 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 to give a final global minimum solution.

Other algorithm 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 needs 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 [6]. In 1969 the branch and bound algorithm [5] was one of the most well known algorithms for solving complex models like the travelling salesman problem [5]. However, this algorithm was restricted to specific types of problems like concave minimization and when computing worst case, its time complexity was exponential.

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 [1, 6]. Various sub-categories of stochastic algorithms have appeared, like probabilistic approaches, Monte Carlo approaches, evolutionary algorithms [1] and metaheuristics methods [3]. 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), differential evolution (metaheuristic method), genetic algorithm (evolutionary algorithm), cross-entropy method (Monte-Carlo algorithm), and simulated annealing (probabilistic algorithm) [1].

Global Search

Global Search (GS) is hybrid heuristic algorithm that generates population points using the scatter search algorithm [**?**] 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 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

$$|x_i - b_j| > dr_j \tag{2.10}$$

  where x_i is the ith trial point and b_j is the jth basin of attraction centre; d is the distance threshold factor, default value 0.75, and r_j is the radius of jth 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

$$|xc_k - xc_i| > T_x \max(1, |xc_i|) \tag{2.11}$$

$$or \tag{2.12}$$

$$|fc_k - fc_i| > T_f \max(1, |fc_i|) \tag{2.13}$$

where xc_k and xc_i is every kth convergent point and the convergent point for the ith trial point; fc_k and fc_i is every kth objective function value and objective function for the convergent point for the ith trial point; T_x and T_f are the x tolerance and function tolerance, default 1 10^-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

$$l = l + p_f(1 + |l|) \tag{2.14}$$

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 [**?**]. 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

$$v_{i,j}^{k+1} = v_{i,j}^{k} + c_1 r_1 (x_b^k - x_{i,j}^k) + c_2 r_2 (x_g^k - x_{i,j}^k) \tag{2.15}$$

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; 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

$$x_{i,j}^{k+1} = x_{i,j}^{k} + v_{i,j}^{k+1} \tag{2.16}$$

using the velocity obtained by Equation 2.15. To avoid premature convergence on local minima or over exploration of the particles Shi and Eberhart [] 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

$$v_{i,j}^{k+1} = w v_{i,j}^{k} + c_1 r_1 (x_{local}^k - x_{i,j}^k) + c_2 r_2 (x_{global}^k - x_{i,j}^k). \tag{2.17}$$

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 2.15 to

$$v_{i,j}^{k+1} = \chi\left[v_{i,j}^{k} + c_1 r_1 (x_b^k - x_{i,j}^k) + c_2 r_2 (x_g^k - x_{i,j}^k)\right] \tag{2.18}$$

$$\chi = \frac{2}{/}\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right| \; where \; \phi = c_1 + c_2, \; \phi > 4, \tag{2.19}$$

this equation also prevents particle divergence. The schematic movement of the particle shown in Figure 2.2 follows the Equation 2.18.

When compared to the inertia weight Equation 2.15, 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 developed by Krink et al. [],. One variant of PSO known Global Convergence PSO, used to solve the crystal structure prediction problem discussed in Chapter 4. Van den Bergh and Engelbrecht [] developed *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 address stagnation by randomly searching around the global best particle in an adaptive radius at each iteration. The GCPSO uses the align 2.17 and 2.18 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

$$v_{i_g}^{k+1} = -x_{i_g}^k + x_{i_g}^k + w^k v_{i_g}^k + \rho^k (1 - 2r_3^k) \tag{2.20}$$

$$\tag{2.21}$$

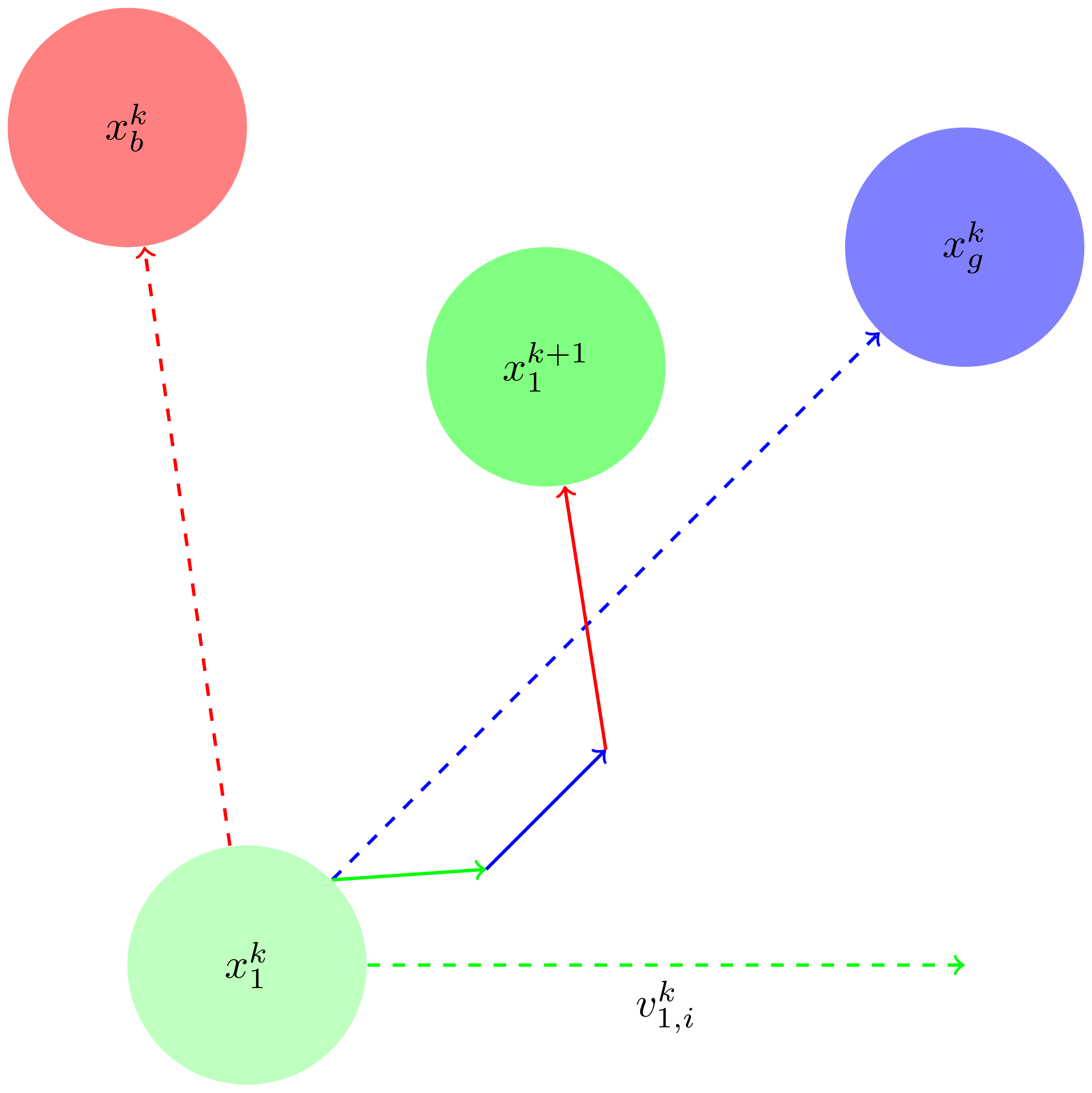


Figure 2.2: Schematic of the particles movement using Equation 2.17

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

$$\rho^{k+1} = \begin{cases} 2\rho^k, & \text{if} \check{s}^{k+1} > s_c, \\ \frac{1}{2}\rho^k, & \text{if} \check{a}^{k+1} > a_c \\ \rho^k, & \text{otherwise} \end{cases} \tag{2.22}$$

where s_c is the success threshold and a_c is the failure threshold. A success is indicated when Equation 2.20 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

Compute farm

As discussed in Chapter 2 global optimization solvers depend on the computational time, resources, and fault tolerant black box functions. 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 a lot of resources to run the model because the population based algorithms each point in the population would still need to be ran in serial to prevent any resource contention if the computer is unable to sustain multiple greedy resource function evaluations. 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 can be evaluated in parallel. Another feature of computefarm that handles global optimization function dependencies is the risk of failure. If a client disconnects or fails to run the function evaluations, computefarm handles the failure by reassigning the evaluation to another client in hopes of success. This is more beneficial than on a single machine, a failure can cause the whole optimization to stop and to restart again. In some optimization case this be months of computational time lost to a failure. 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 direct by the server machine. For global optimization this means the server delegates points in the search space to client computers to evaluate with the black box function. The client then returns the result to the server machines and waits on further positions to evaluate. The server then a provides a list of results to the global optimization solver from each point evaluated. In the case of a failure the server will receive a failed return value from the client and will place 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 ensure fault tolerance in the computefarm software such that the global optimization solver does not need to restart if a failure occurs. By doing this the parallelization of evaluating multiple black functions is encapsulated in fault tolerate software known as computefarm that allows for parallelization without resource contention occurring.

Chapter 4

Applications

Chapter 5

Conclusion

I conclude that I have solved the problem!

References

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

[2] 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.

[3] 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.

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

[5] L. Liberti. Introduction to global optimization. 2000.

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

[7] 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.