PARTICLE SWARM OPTIMIZATION FOR ENERGY MINIMIZATION OF MOLECULAR SYSTEMS

by

David A. Hickman

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Masters of Science (Applied Mathematics and Statistics).

Golden, Colorado

Date ______________________

Signed: ______________________
David A. Hickman

Signed: ______________________
Dr. Stephen Pankavich
Thesis Advisor

Golden, Colorado

Date ______________________

Signed: ______________________
Dr. Willy Hereman
Professor and Department Head
Department of Applied Mathematics and Statistics

ABSTRACT

The minimization of a potential energy function can be used to provide insight into the ground state configuration of a wide range of molecular systems. Optimization problems of this type can be challenging for deterministic (e.g. line search) optimization algorithms due to the size of the search space and the large number of local minima that are inherent within molecular configuration problems. We describe how Particle Swarm Optimization, a stochastic optimization algorithm inspired by flocking behavior, can be used to accurately and efficiently solve energy minimization problems associated with molecular systems.

TABLE OF CONTENTS

ABSTRACT iii

LIST OF FIGURES vi

LIST OF TABLES vii

LIST OF SYMBOLS viii

CHAPTER 1 INTRODUCTION 1

1.1 Optimization 3

1.1.1 Deterministic Algorithms 5

1.1.2 Stochastic Algorithms 5

CHAPTER 2 PARTICLE SWARM OPTIMIZATION 7

2.1 Swarm Intelligence 7

2.2 Particle Swarm Optimization 9

2.2.1 Global Best PSO 10

2.2.2 Local Best PSO 11

2.3 PSO Characteristics 12

2.3.1 Velocity Clamping 13

2.3.2 Parameters 14

2.3.3 Social Network Structures 15

2.3.4 Termination Conditions 18

CHAPTER 3 NONLINEAR MOLECULAR MODEL 20

3.1 Bond Potential 20

3.2 Angle Potential 21
3.3 Van der Waals Potential 21
CHAPTER 4 IMPLEMENTATION 24
4.1 Serial Implementation 24
4.2 Parallel Implementation 25
CHAPTER 5 RESULTS 27
5.1 Parameter Selection 27
5.2 Minimal Energy Configuration 31
5.3 Comparisons 31
5.4 Conclusions 34
5.5 Moving Forward 35
REFERENCES CITED 36
APPENDIX - FORTRAN CODE 38
A.1 Potential Function 38
A.2 Serial PSO 40
A.3 Parallel PSO 45

LIST OF FIGURES

Figure 1.1 Global and local minima 4

Figure 2.1 Social Network Structures 17

Figure 3.1 Bonded Atoms 20

Figure 3.2 Bond Angle 21

Figure 3.3 12-6 Lennard-Jones Potential 22

Figure 3.4 Non-Bonded Potential 23

Figure 3.5 Nonlinear Molecular System 23

Figure 5.1 Energy Histogram with $c_2 = 0.90$ 29

Figure 5.2 Energy Histogram with $c_1 = 0.30$ 30

Figure 5.3 Minimal Energy Configuration 31

Figure 5.4 Rastrigrin function in 2D 33

LIST OF TABLES

Table 4.1 PSO Settings 25

Table 5.1 Parameter Sweep 28

Table 5.2 Performance Measure 28

Table 5.3 Optimization Method Comparison 32

Table 5.4 Rastrigrin Comparison 34

LIST OF SYMBOLS

inertial parameter	ω
cognitive parameter	c_1
social parameter	c_2
uniformly distributed random variables	r_1, r_2
maximum velocity	V_{max}
particle position	$\mathbf{x}_i$
personal best position	$\mathbf{p}_i$
global best position	$\mathbf{g}$
local best position	$\mathbf{g}_i$

CHAPTER 1

INTRODUCTION

Molecular structure has become an important area of research in a number of diverse fields ranging from materials science to computational chemistry and has provide us with insight into such phenomenon as protein folding, macromolecule dynamics, and cell-mediated processes. Studies of molecular structure typically begin with a molecular model describing the interactions between the atoms within the molecule. These interatomic interactions can be formulated in terms of classical mechanics principles. Here the atoms interact via conservative forces which can be expressed in terms of a potential energy function V. There are several applications one can pursue using this Newtonian formalism. One application is the so-called atomic conformation problem. The aim here is to find the lowest energy state of the molecule so as to determine its most likely configuration in the absence of external forces. In nature, isolated systems tend towards the minimum energy state, so finding the global minimum of the potential energy should provided insight into the molecular structure. Hence, the atomic conformation problem can be reformulated in terms of an optimization problem and is often referred to as the energy minimization problem.

Energy minimization techniques are often used in conjunction with other molecular modeling applications; for example, consider the molecular dynamics (MD) simulations. This application begins by assuming the form of a potential energy function V and using it to define a Hamiltonian

$$H = \sum_{i=1}^{N} \frac{p_i^2}{2m_i} + V(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N), \tag{1.1}$$

where N is the number of atoms in the molecule and m_i, $\mathbf{q}_i$, and $\mathbf{p}_i$ are the mass, position, and momentum of atom i, respectively.

The Hamiltonian can then be used to generate a system of ODE's given by

$$\left.\begin{aligned} \dot{\mathbf{p}}_i = & \quad -\frac{\partial H}{\partial \mathbf{q}_i} \\ \dot{\mathbf{q}}_i = & \quad \frac{\partial H}{\partial \mathbf{p}_i}. \end{aligned}\right\} \quad \text{Hamilton's equations} \tag{1.2}$$

Often, such large dimensional systems of ODE's can be solved numerically, using a finite difference method for example, for the motion of the atoms within the molecule of interest. This approach allows one to view the pathway to energy minimization from a given initial configuration. The initial configuration is of the utmost importance because for large systems an MD simulation can become computationally intractable and can only simulate the dynamics for a few nanoseconds. The initial configuration used is commonly determined using experimental techniques such as x-ray crystallography. However, there can be interactions between the molecule of interest and the surrounding crystal which lead to distortions of the recorded molecular configuration. These distortions can lead to unphysical energy changes during the simulation. To overcome this an energy minimization is performed on the initial configuration before the simulation is run. Hence, energy minimization is an important step before running MD simulations. Alternatively, if one is interested in determining only molecular structure, and not the pathway to energy minimization MD simulations are unnecessary and one can utilize an optimization method instead.

The main focus of this work will demonstrate how the Particle Swarm Optimization algorithm can be used to determine a molecule's structure so that the potential energy is minimized. Section 1.1 provides a formal definition of the optimization problem. In Chapter 2 we give a detailed account of the Particle Swarm Optimization algorithm. The representative molecular model under consideration is described in Chapter 3, and in Chapter 4 we give details on how PSO is implemented. Finally, in Chapter 5 we report our results are reported and comparisons are made with other optimization methods.

1.1 Optimization

In applied mathematics optimization refers to the procedures used to find a "best" solution - amongst a set of candidate solutions - to a given problem. In general there are three aspects that are common to all optimization problems: an objective function, a search space, and constraints. The objective function is the quantity in which one wishes to optimize. The search space S is the set containing the candidate solutions which are used to evaluate the objective function. Constraints limit the search to candidate solutions contained within some feasible space $F \subseteq S$. Constraints are often found in real world optimization problems where time, money, and resources are limited.

Solutions to optimization problems are commonly referred to as global and local optima. In the case of minimization, we have global and local minima which are defined below and illustrated in Figure 1.1.

Definition 1.1 *The solution* $\mathbf{x}^* \in F$, *is a global minimum of an objective function* f, *if* $f(\mathbf{x}^*) < f(\mathbf{x})$, $\forall \mathbf{x} \in F$ where $F \subseteq S$.

Definition 1.2 *A solution* $\mathbf{x}^* \in N \subseteq F$, *is a local minimum of an objective function* f, *if* $f(\mathbf{x}^*) < f(\mathbf{x})$, $\forall \mathbf{x} \in \mathcal{N}$ *where* $\mathcal{N} \subseteq F$ *is a set of feasible solutions in the neighborhood of* $\mathbf{x}^*$.

With the above definitions, one might assume that the only way to determine if a point was in fact a solution would be to test all the surrounding points to ensure none of them have a smaller function value. However, if the objective function is twice continuously differentiable then we have some analytical tools at our disposal to determine whether or not a point is indeed a local minimum.

Our first two tools provide us with necessary conditions regarding local minima:

Theorem 1.1 *If* $\mathbf{x}^*$ *is a local minimum and* f *is continuously differentiable in an open neighborhood of* $\mathbf{x}^*$*, then* $\nabla f(\mathbf{x}^*) = 0$.

Theorem 1.2 *If* $\mathbf{x}^*$ *is a local minimum of* f *and the Hessian matrix* $\nabla^2 f$ *exits and is continuous in an open neighborhood of* $\mathbf{x}^*$*, then* $\nabla f(\mathbf{x}^*) = 0$ *and* $\nabla^2 f(\mathbf{x}^*)$ *is positive semidefinite.*

Our final tool provides us with sufficient conditions on the derivatives of the objective function that guarantee that $\mathbf{x}^*$ is a local minimum:

Theorem 1.3 *Suppose* $\nabla^2 f$ *is continuous in an open neighborhood of* $\mathbf{x}^*$ *and that* $\nabla f(\mathbf{x}^*) = 0$ *and* $\nabla^2 f(\mathbf{x}^*)$ *is positive semidefinite, then* $\mathbf{x}^*$ *is a local minimum of* f*.*

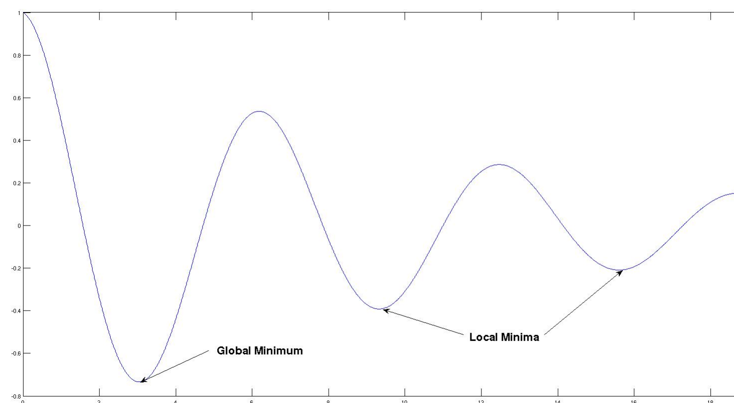


Figure 1.1: Global and local minima

These theoretical results provide the foundation upon which the majority of optimization algorithms are built. For instance, most deterministic methods, in one form or another, attempt to find a point $\mathbf{x}^*$ where $\nabla f(\mathbf{x}^*) = 0$. In the following subsection we provide a brief overview of the most common optimization paradigms, categorized into deterministic and stochastic algorithms.

1.1.1 Deterministic Algorithms

Deterministic algorithms typically begin by requiring the user to specify some starting point $\mathbf{x}_0$ and then the algorithm generates a sequence of points $\{\mathbf{x}_k\}_{k=0}^{N}$ until it terminates either because a solution is found within some specified tolerance or a maximum number of iterations has been exceeded. The manner in which the sequence of points is generated is what distinguishes one deterministic algorithm from another. Some of the more common deterministic algorithms are the so-called line search methods. These approaches begin by selecting a search direction $\mathbf{z}_k$ and then they generate a new candidate $\mathbf{x}_{k+1} = \mathbf{x}_k + \gamma\mathbf{p}_k$, where $\gamma > 0$ is referred to as the step size. The optimal step can be determined by solving the one-dimensional optimization problem

$$\min \phi(\gamma) = f(\mathbf{x}_k + \gamma\mathbf{p}_k). \tag{1.3}$$

Solving (1.3) exactly would derive the maximum benefit from the search direction $\mathbf{p}_k$, but in many cases is computationally expensive. Thus, line search methods typically generate a set of trial step sizes until an approximate solution to (1.3) is found. At the new point $\mathbf{x}_{k+1}$ a new search direction is determined and the procedure is repeated N times. As one might expect, the steepest descent direction $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)/||\nabla f(\mathbf{x}_k)||$ is the most widely used search direction. Two algorithms that use the gradient of the objective function to determine the search direction are the *gradient descent* and *conjugate gradient* algorithms. We will investigate the use of the conjugate gradient method in an energy minimization problem in section 5.2

1.1.2 Stochastic Algorithms

Stochastic algorithms, as the name suggests, introduce some form of randomness into the search for a solution to the optimization problem. The manner in which randomness is used depends on the specific algorithm. For example, consider the *evolutionary algorithm* (EA). This algorithm utilizes mechanisms inspired by Darwin's theory of natural selection. These mechanisms include selection, reproduction, and mutation. Candidate solutions are repre-

sented by members of a population. The objective function serves as the selection process by determining the quality of a candidate solution. If a candidate solution satisfies a certain criteria it is allowed to reproduce with other members who have met the criteria. Candidate solutions who are not selected die off. The offspring of the fittest individuals are then subject to a random mutation and become the next generation of candidate solutions. The process is then repeated. The stochastic element is introduce in the (EA) through randomly distributing the population throughout the search space and in the mutation operator.

Algorithm 1 Evolutionary Algorithm [2]

Let $t = 0$ be a generation counter;
Initialize a population $P(0)$ in an N-dimensional search space
repeat
  Evaluate the objective function, $f(\mathbf{x}_i)$, of each individual, $\mathbf{x}_i$, in the population, $P(t)$;
  Perform reproduction to produce offspring;
  Perform mutation on offspring;
  Select population $P(t+1)$ of new generation;
  Advance to the new generation, i.e. $t = t + 1$;
until termination condition is satisfied;

There are many stochastic algorithms that have been developed. In general, they perform well for global optimization problems since their search is not based on the derivatives of the objective function, which makes these algorithms less likely to become trapped in local minima. Particle Swarm Optimization (PSO) is one such algorithm which is the main focus of this work. PSO's inner workings and characteristics are described in detail in the next chapter.

CHAPTER 2
PARTICLE SWARM OPTIMIZATION

Since its inception Particle Swarm Optimization has been used on a variety of different optimization problems ranging from neural network training, linear antenna arrays, and power/voltage control for utility companies [3, 7, 8]. PSO is rooted in a branch of artificial intelligence called swarm intelligence. Before we begin an in-depth discussion on how PSO executes an optimization procedure, we present some background on swarm intelligence and how it inspired the development of PSO.

2.1 Swarm Intelligence

The term Swarm Intelligence or SI was first introduced by Beni and Wang in 1993 as a way to describe the "intelligent" behavior of cellar robotic systems [1]. Since then SI has been used to describe the property of a system, natural or artificial, whereby the collective behaviors of (unsophisticated) entities interacting locally with their environment cause coherent functional global patterns to emerge [11]. There are many examples in nature where SI has been observed. Termite mounds, that are riddled with a complex system of interconnected tunnels, have been recorded to reach 30 meters in diameter. The foraging behavior of ants that emerges from the release of pheromones is another example. Flocks of birds and schools of fish organize themselves in dynamically intricate spatial patterns to throw off predators. These examples, and others like them, result from the interactions between members of the swarm. The actions of a single member of the swarm is not necessarily complex, but taken collectively, these actions produce complex structures that are not readily deduced from the actions of the individual.

In recent decades attempts at modeling SI, and in particular flocking behavior, has attracted the interest of computer scientists and mathematicians alike. C.W. Reynolds was able to demonstrate through visual simulations that flocking is an emergent behavior that

arose from the individual birds following three simple rules: collision avoidance, velocity matching, and flock centering [12]. Collision avoidance was implemented so that members of the flock will maintain some degree of spatial separation. Similar to collision avoidance, velocity matching ensures that members of the flock have similar trajectories to their closest neighbors. Finally, flock centering was implemented so that nearest neighbors would remain close to one another as the flock moves.

The simulations of Reynolds motivated two researchers, Eberhart and Kennedy, to develop a simplified social model based on nearest neighbors and velocity matching. Their motivation, much like Reynolds, was to graphically simulate the graceful but unpredictable choreography of a bird flock [5]. Within Eberhart and Kennedy's model, the position of each bird was randomly distributed on a torus pixel grid. Each bird also performed velocity matching of its nearest neighbor. The end result was synchronized movement that rapidly faded to the flock flying in one direction. Random adjustments to velocities, which Eberhart and Kennedy dubbed "craziness", were added in an attempt to circumvent this. Their model was expanded further with the addition of the "cornfield" which was a 2-dimensional plane that the birds were flown through. After each iteration of the simulation, each bird would evaluate an objective function based on its position in the "cornfield". Each bird kept track of the best position it had encountered based upon function evaluations using its coordinates in the "cornfield"; this position was referred to as *pBest* which is short for personal best position. Eberhart and Kennedy also include memory of the best position that the entire flock as a whole had encountered. This location they labeled the global best position or simply *gBest*. *pBest* and *gBest* were used to dynamically update the location of the birds, and after a few iterations the flock would come to rest in the global best position of the "cornfield" which, as it so happened, was the global minimum of the objective function. Eberhart and Kennedy had unknowingly created an optimization algorithm which they called Particle Swarm Optimization. Individuals were referred to as particles because they had zero mass and volume despite the fact that each individual had a position and

velocity vector. The term swarm was used because the algorithm adheres to the following principles of SI as defined by Millonas [9]:

- **Proximity principle:** the group of individuals should be able to carry out simple space and time computations
- **Quality principle:** the group of individuals should be able to respond to quality factors of the environment
- **Principle of diverse response:** the group of individuals should not commit its activities along excessively narrow channels
- **Principle of stability:** the group of individuals should not change its mode of behavior every time the environment changes
- **Principle of adaptability:** the group of individuals must be able to change its behavior mode when it is worth the computational cost

The particles evaluate the objective function, over a series of time steps, using their positions within an n-dimensional search space which meets the criteria of the proximity principle. The quality principle is implemented through the use of the *pBest* and *gBest* locations. Allocation of responses between *pBest* and *gBest* ensure a diversity of response. Furthermore the state of the swarm changes only after *pBest* and *gBest* change thus providing stability. Finally, the state of the swarm changes only when *pBest* and *gBest* change which implies adaptability.

2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population based stochastic search optimization algorithm. The population or swarm consists of a number of particles N that explore a n-dimensional search space where the location of each particle within the space is adjusted based on its own experience and that of its neighbors.

Let $\mathbf{x}_i(t), \mathbf{v}_i(t) \in R^n$ for $i = 1, \ldots, N$ be the position and velocity of the *ith* particle, respectively. Here $t \in \mathbb{N}$ denotes a discrete time step. The position of the *ith* particle is adjusted by adding the velocity to the current position, that is

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1). \tag{2.1}$$

In practice, the initial positions are confined within some boundary i.e.

$$\mathbf{x}_{min} \leq \mathbf{x}_i(0) \leq \mathbf{x}_{max}.$$

There are two main variations of the PSO algorithm, namely the global best PSO and the local best PSO, which differ only in the size of their neighborhoods. The social interconnectivity of the swarm is often referred to as the neighborhood topology or social network structure. These two variants of PSO will be described in detail in the following sections

2.2.1 Global Best PSO

In the global best PSO, or simply *gbest* PSO, the neighborhood of each particle is the entire swarm. The neighborhood topology of *gbest* PSO is a star topology depicted in Figure 2.1(a). Every particle experiences an attraction towards its personal best value *pBest* and the best position encountered by any particle which we call the global best or simply *gBest*. Let the personal best position of particle i, in a n-dimensional search space be given by

$$\mathbf{p}_i = [p_{i1}, p_{i2}, \ldots, p_{in}],$$

and let the global best position be given by

$$\mathbf{g} = [g_1, g_2, \ldots, g_n].$$

After each iteration t the velocity of the *ith* particle is updated using the following equation:

$$v_{ij}(t+1) = \omega v_{ij}(t) + r_1 c_1 (p_{ij} - x_{ij}(t)) + r_2 c_2 (g_j - x_{ij}(t)), \tag{2.2}$$

where r_1 and r_2 are random numbers uniformly distributed between 0 and 1, p_{ij} and g_j are the *jth* component of pBest and gBest, respectively, c_1 (cognitive parameter) and c_2 (social

parameter) are weights that scale the attraction towards p_{ij} and g_j, while ω is referred to as the inertial parameter. The particle positions are updated using (2.1) then each particle evaluates the objective function f using its current position i.e. $f_i = f(\mathbf{x}_i(t+1))$ and the vectors $\mathbf{p}_i$ and $\mathbf{g}$ are updated in the following way:

$$\mathbf{p}_i(t+1) = \begin{cases} \mathbf{p}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{p}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{p}_i(t)) \end{cases} \tag{2.3}$$

$$\mathbf{g} = \mathbf{p}_i(t+1) \text{ s.t. } f(\mathbf{p}_i(t+1)) \leq f(\mathbf{p}_j(t+1)) \text{ for } j = 1, \ldots, N. \tag{2.4}$$

Algorithm 2 Global Best PSO

Step I Initialize the swarm
Here we choose the swarm size S, that is we choose how many particles will be created. The particles are then randomly distributed throughout the search space subject to the boundaries $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$.

Step II Initialize $\mathbf{p}_i$ and $\mathbf{g}$
Typically we set $\mathbf{p}_i = \mathbf{x}_i(0)$ since at $t = 0$ a particle's starting position and its previous best position coincide. Next we set $\mathbf{g} = \mathbf{x}_i(0)$ such that $f(\mathbf{x}_i(0)) \leq f(\mathbf{x}_j(0))$ for $j = 1, \ldots, S$.

Step III While termination conditions are not met

1. Update $\mathbf{v}_i$ using (2.2)
2. Update $\mathbf{x}_i$ using (2.1)
3. Update $\mathbf{p}_i$ and $\mathbf{g}$ using (2.3) and (2.4)

2.2.2 Local Best PSO

In the local best PSO (lbest PSO) the neighbor topology is the ring structure depicted in Figure 2.1(b). Here the particles share their experiences within a smaller neighborhood rather than communicating their knowledge with the entire swarm. The neighborhood of particle i is defined as

$$\mathcal{N}_i = \{\mathbf{x}_{i-k}, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+k}\} \tag{2.5}$$

where k is the size of the neighborhood. Neighborhoods are based on index only and not on the spatial separation between particles.

There are two primary reasons why neighborhoods based on particle index are preferred [2]:

1. Basing neighborhoods on particle separation would require the calculation of the Euclidean distance between all the particles in the swarm. Evaluating the objective function may already be computationally expensive. Calculating the Euclidean distance would add additional complexity.

2. Neighborhoods based on particle index support the spread of information regarding promising regions to all particles regardless of their current location within the search space.

Not surprisingly, for this variant of PSO we have a local best rather than a global best position which we define as

$$\mathbf{g}_i(t+1) \in \{\mathcal{N}_i |\, f(\mathbf{g}_i(t+1)) \leq f(\mathbf{x}),\, \forall \mathbf{x} \in \mathcal{N}_i\}. \tag{2.6}$$

The velocity is updated by substituting $\mathbf{g}_i$ for $\mathbf{g}$ into equation (2.2). Particle position and personal best positions in lbest PSO are updated in exactly the same way for gbest PSO. Regarding convergence towards the global minimum, research has revealed two main difference between gbest and lbest PSO [6]:

- Because particles share information with the entire swarm, gbest PSO tends to converge faster than lbest PSO but can potentially get trapped in local minimum.

- lbest PSO promotes a larger swarm diversity since particles only have local knowledge of their neighborhood. This makes the particles less likely to get trapped within local minimum but at the cost of slower convergence. The swarm will eventually converge on a single point within the search space since particle neighborhoods overlap.

2.3 PSO Characteristics

The previous section introduced two variations of PSO namely the global best PSO and local best PSO. In what follows we will discuss their various characteristics. Section 2.3.1 will

introduce velocity clamping and why it is necessary. In section 2.3.2 we explore the parameters ω, c_1, and c_2 and their effects on the velocity equation. Section 2.3.3 expands further on the concept of neighborhood topologies and describes several social network structures. Finally in section 2.3.4 we consider typical termination conditions.

2.3.1 Velocity Clamping

During the early stages of development, PSO suffered from swarm explosion, that is the unabated increase of the particle velocities which resulted in swarm divergence. The deficiency was corrected by placing upper and lower bounds on the velocity equation (add citation). Typically, the upper and lower bounds on (1.2) are set to be some fraction of the domain for each dimension in the search space i.e.

$$V_{max,j} = \delta(x_{max,j} - x_{min,j}),$$

where $\delta \in (0, 1]$ and $x_{max,j}$ and $x_{min,j}$ are the maximum and minimum values for $\mathbf{x}(t)$ in dimension $j.V_{max,j}$ is used to adjust particle velocities in the following way:

$$v_{ij}(t+1) = \begin{cases} v_{ij}(t+1) & \text{if } |v_{ij}(t+1)| < V_{max,j} \\ V_{max,j} & \text{if } |v_{ij}(t+1)| \geq V_{max,j}. \end{cases} \tag{2.7}$$

Here, $V_{max,j}$ controls the magnitude of $\mathbf{v}_i(t+1)$. Large values of $V_{max,j}$ allow the particles to explore farther reaches of the search space while smaller values of $V_{max,j}$ allow for exploitation, which is the ability of the algorithm to refine the search around promising areas. The are two important factors concerning velocity clamping of which one should be aware of:

1. Velocity clamping not only places an upper bound on $||\mathbf{v}_i(t+1)||_2$ but it also influences the orientation of $\mathbf{v}_i(t+1)$.

2. It is possible that over the course of several time steps all velocities could become equal to $V_{max,j}$ and eventually all particles would be pressed against the boundary of the search space. In order to overcome this difficulty, researchers introduced the inertial parameter ω.

We will discuss ω, and the other parameters, in the next section.

2.3.2 Parameters

The addition of velocity clamping solved the issue of swarm explosion, however, it was soon observed after its introduction that the particles were unable to converge towards the global minimum. Instead of convergence the particles would oscillate on wide trajectories around their best positions. Research revealed that convergence failure was due to the inability to control particle velocities; if the previous velocity term of a particle is to large, then the particle will fly by the best position and it will never settle down. Thus the previous velocity term needed to decrease each time step, and so the inertial parameter ω was introduced [13]. The inertial parameter, as one might expect, plays a crucial role in the algorithm's ability to converge on a good solution. If we set $\omega \geq 1$ then particle velocities increase over time and will eventually reach $V_{max,j}$ resulting in swarm divergence. If one were to set $\omega < 1$, then particle velocities will eventually tend to zero and the particles will stop moving. Large values of ω allow for exploration of the search space while smaller values of ω allow for convergences onto promising solutions. Care must be taken, however, because if the inertial parameter is too small then particles have very little momentum meaning their directions can change dramatically, and the end result being that they can be quickly dragged into local minima. In practice the inertia parameter is set to a value slightly large than 1, typically 1.2, then it is decreased at each time step t. A common scheme used to dynamically adjust ω is given below:

$$\omega(t) = (\omega(0) - \omega(T))\frac{(T-t)}{T} + \omega(T), \tag{2.8}$$

where T is the maximum number of iterations, $\omega(0)$ and $\omega(T)$, are the initial and final values, respectively. This scheme is particularly effective at balancing exploration during the initial search and exploitation near the final time steps. Alternatively, random adjustments and nonlinear decreasing schemes have been used to dynamically adjust the inertial parameter.

The cognitive and social components direct the particles towards their own personal best positions $\mathbf{p}_i$ and the global best position $\mathbf{g}$ in the case of gbest PSO and local best

position $\mathbf{g}_i$ in the case of lbest PSO. During the search, the cognitive and social parameters, c_1 and c_2, greatly impact the exploration capabilities of the PSO algorithm. For example, setting $c_1 < c_2$, the particles are attracted more towards the global best position rather than their individual best positions while setting $c_1 > c_2$, the particles are biased towards their respective best positions. Setting these parameters to relatively small values (typically less than 1) results in smooth particle trajectories, that is to say that the position of the particles do not change rapidly during the search. Setting c_1 and c_2 to values larger than 1, however, tends to produce more abrupt changes in particle positions. Along with the trajectories the particles take, the cognitive and social parameters affect the depth of the search in the domain space since these parameters scale the velocity equation.

The selection of the parameters $\omega, c_1,$ and c_2 dramatically impacts the performance of the PSO. Parameter choice depends heavily on the problem and its dimensionality. Determining the optimal parameters for a given problem is known in the literature as meta-optimization. There has been an attempt by Miessner et al. [8] to find the optimal parameters by using another instance of the PSO as a meta-optimizer. More typical methods used to find the optimal parameters involve running the algorithm several times for a given set of parameters and recording if the algorithm managed to locate the global minimum and if it did also recording the number of function evaluation the PSO had to perform to reach the global minimum. An average is computed and the process is repeated again for another set of parameters. Thus through comparison, a reliable set of parameters is found. Naturally, the parameters will differ depending on the problem to be optimized.

2.3.3 Social Network Structures

The social network structure or neighborhood topology describes the way in which information flows throughout the swarm. A social network structure is composed of overlapping neighborhoods in which particles only share information with those in their neighborhood. Particles tend to be drawn to the most "successful" member of their group which is analogous to many biological systems where the most successful individuals have a greater influence

on those around them. In terms of PSO, the most successful particle becomes the local best position $\mathbf{g}_i$.

Performance is heavily influenced by the social network structure. There are three aspects of the social network structure that effect how information flows through out the swarm:

1. The interconnectivity of the nodes (particles) of the network.
2. The degree of clustering (clustering occurs when a nodes neighbors are also neighbors with one another).
3. The average shortest distance between nodes.

Information that is shared with most of the swarm represents a highly interconnected social structure. In this environment, the perceived best particle is quickly disseminated throughout the swarm which typically leads to faster convergence compared to less connected social structures. There is a tradeoff however, because faster convergence can lead to suboptimal solutions i.e. local minima. Less interconnected structures tend to provide better coverage through out the search space.

Some of the most widely studied social network structures are listed below:

- The star topology, depicted in Figure 2.1(a), is where all the particles are connected to one another. This is the topology used the gbest PSO described earlier. This topology is best suited for unimodal problems, that is, the objective function has only one minimum.
- The ring topology depicted in Figure 2.1(b). In this structure the particles only communicate with their nearest neighbors on the ring. A particle's neighbors form its neighborhood as defined by (5). Information flows slower in this structure compared to the star topology, but larger areas of the search space are explored. With regards to multimodal problems where many minima exist, the ring topology often provides better solutions than those obtained using the star topology.

- The Von Neumann topology depicted in Figure 2.1(c). Here particles are connected in a grid like structure. Several investigations have shown that the Von Neumann structure outperforms other structures in a large number of problems [6, 10].
- The four clusters topology depicted in Figure 2.1(d). This structure is composed of four clusters where a single cluster is connected to two others. Within each cluster are five interconnected neighbors.

As is so often the case, the optimal network topology is problem dependent. Unimodal problems are best served using highly interconnected structures like the star while multimodal environments are often best treated using structures that are less interconnected. In our use of PSO to solve an energy minimization problem (Chapter 3), the star topology is utilized.

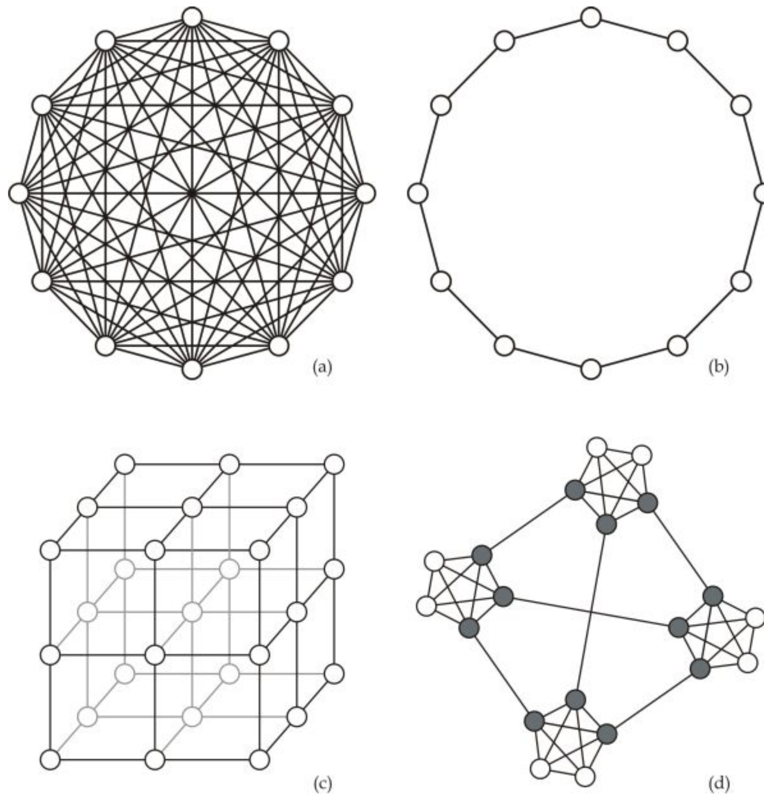


Figure 2.1: Social Network Structures

2.3.4 Termination Conditions

Termination conditions are an important aspect of the PSO algorithm. Commonly used termination conditions are given below:

- *Terminate when the algorithm has exceeded a maximum number of iterations.* Care must be taken when choosing a maximum number of iterations since choosing to few iterations and the algorithm will terminate prematurely before an optimal solution can be found.

- *Terminate when an acceptable solution has been found.* Suppose $\hat{\mathbf{x}}$ is the global minimum of an objective function f and let $\epsilon > 0$ be given. Then $\mathbf{x}$ is an acceptable solution if $|f(\hat{\mathbf{x}}) - f(\mathbf{x})| < \epsilon$. This termination requires the solution to be known a priori and is widely used as method to test an algorithm using benchmarks like Rastigrin's function.

- *Terminate if no improvement has been made after a set number of iterations.* There are several ways in which to measure improvement. For example, one could monitor the position of $\mathbf{g}$. If this positions does not change for a given number of iterations then the algorithm can be terminated. Another measure of improvement is monitor the average particle velocity. If the average velocity is relatively small then the algorithm has little chance for improvement since the particles are barely moving.

- *Terminate when the swarm radius has collapsed.* Let the center of the swarm be defined as

$$\rho = \frac{1}{S} \sum_{i=1}^{S} \mathbf{x}_i, \tag{2.9}$$

where S is the number of particles in the swarm. Then the swarm radius is given by

$$r = ||\mathbf{x}_i - \rho|| \text{ s.t. } ||\mathbf{x}_i - \rho|| \geq ||\mathbf{x}_j - \rho|| \text{ for } j = 1, \ldots, S. \tag{2.10}$$

The algorithm will be terminated when $r < \epsilon$. If ϵ is to large then the algorithm will terminate prematurely before a good solution can be found. If ϵ is to small,

however, then the algorithm may perform unnecessary iterations without any sign of improvement.

- *Terminate when the slope of the objective function* f *is approximately zero.* Considering the expression

$$f^{'}(t) = \frac{f(\mathbf{g}(t)) - f(\mathbf{g}(t-1))}{f(\mathbf{g}(\mathrm{t}))} \tag{2.11}$$

  as a means of approximating the slope of the objective function. If $f < \epsilon$ for a consecutive number of time steps, then this implies that the swarm has converged onto $\mathbf{g}(t)$. This termination condition is considered to be superior to the other methods above because the termination criteria is based the search space itself. It does, however, have the shortcoming that the search can be prematurely terminated if some of the particles get trapped into local minima while the rest of the swarm is still exploring the search space. This can be overcome by combining this condition with radius criteria so that the whole swarm has converged on the same point before terminating the algorithm. It should also be said that this condition also requires the algorithm to evaluate f which does add additional complexity.

There are two important criteria to keep in mind when selecting termination conditions:

1. The condition should not terminate the search prematurely since only suboptimal solutions may be found.

2. The condition should not add unnecessary computational complexity.

In most cases a combination of the above conditions are employed. In the following section we demonstrate how PSO can be used to predict the structure of a 16-atom molecule by minimizing a potential energy function.

CHAPTER 3

NONLINEAR MOLECULAR MODEL

As discussed in Chapter 1, molecular systems can be modeled using a classical Newtonian framework. Here atoms within a molecule interact via conservative forces which can be reformulated in terms of a potential energy function V. For our purposes, the potential energy can be expressed as the sum of three terms, those arising from bonds between atoms, those representing bond angles, and the Van der Waals potential:

$$V = V_{bond} + V_{angle} + V_{Van\,der\,Waals}. \tag{3.1}$$

Instead of studying the motion of the molecule using MD simulations, we will focus on determining molecular structure only, and hence seek to minimize the potential energy V.

3.1 Bond Potential

Probably the most common potential used to model the attraction between two bonded atoms within a molecule is the harmonic potential. Let $\mathbf{r}_i$ and $\mathbf{r}_j$ be the positions of two bonded atoms (see Figure 3.1). Then the harmonic potential V_{bond} is given by

$$V_{bond} = \frac{1}{2}\sum_{i<j} k_{ij}(r_{ij} - r_0)^2, \tag{3.2}$$

where $r_{ij} = ||\mathbf{r}_i - \mathbf{r}_j||_2$, k_{ij} is a spring constant which equals zero if the atoms are not bonded, and r_0 is the equilibrium distance.

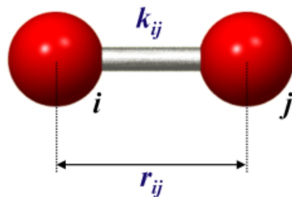


Figure 3.1: Bonded Atoms

3.2 Angle Potential

Within a molecule it is often the case that two bonds can share a common atom as Figure 3.2 illustrates. In such a configuration one can define an angle between two atoms. Let $\mathbf{r}_i$, $\mathbf{r}_j$, and $\mathbf{r}_k$ be the positions of atoms i, j, and k, respectively. Furthermore, let

$$\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j \text{ and } \mathbf{r}_{kj} = \mathbf{r}_k - \mathbf{r}_j.$$

Then the angle θ_{ijk} can be expressed in terms of a cosine

$$\cos\theta_{ijk} = \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{kj}}{r_{ij} r_{kj}}, \tag{3.3}$$

where $\cdot$ denotes an inner product. One widely used angle potential in molecular dynamics (MD) simulations is the cosine harmonic angle potential, V_{angle}, given by

$$V_{angle} = -\sum_{i<j<k} B_{ijk} \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{kj}}{r_{ij} r_{kj}}, \tag{3.4}$$

where B is a tensor of angle interaction strength parameters with entries B_{ijk} that are nonzero if and only if k_{ij} and k_{jk} are both nonzero.

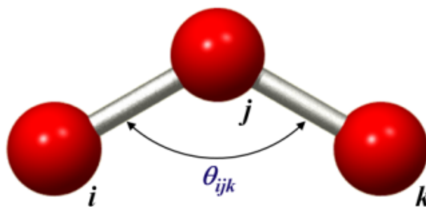


Figure 3.2: Bond Angle

3.3 Van der Waals Potential

The third term in the potential energy expression, $V_{Van\,der\,Waals}$, accounts for the attractive and repulsive forces between non-bonded atoms. Possibly the most widely employed potential used to model the energy of Van der Waals interactions is the 12-6 Lennard-Jones potential

$$V_{LJ} = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] = \epsilon \left[\left(\frac{r_m}{r_{ij}} \right)^{12} - 2 \left(\frac{r_m}{r_{ij}} \right)^{6} \right], \tag{3.5}$$

where ϵ is the depth of the potential well, σ is a finite distance where the potential tends to zero, r_m is the separation at which the potential is at a minimum, and r_{ij} is again the distance between atoms i and j. The 12-6 Lennard-Jones potential tends to zero as the atomic separation increases, but as the atomic separation decreases values of the potential decrease until a minimum is reached, after which the potential increases asymptotically (see Figure 3.3).

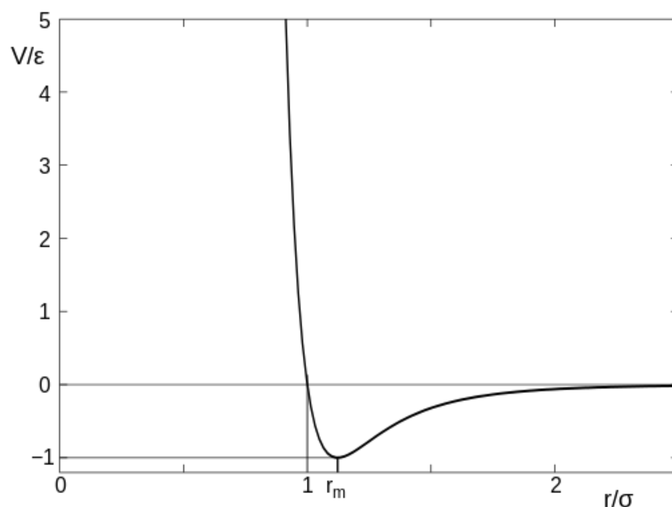


Figure 3.3: 12-6 Lennard-Jones Potential

As mentioned in Chapter 1, MD simulations are computationally expensive for large molecular systems, so researchers have developed a method called "coarse graining" where an N-body system is represented with a reduced number of degrees of freedom. Coarse graining has attracted much attention in the MD community. A recent study by Voth et al. [4] showed how memory minimization can be used as an objective for coarse graining. To demonstrate their idea Voth and his colleagues used a 16 atom molecule (depicted in Figure 3.5) whose energy they modeled using equations (3.2) and (3.4), but rather than using the 12-6 Lennard-Jones potential to model the energy of non-bonded atoms they instead

used the following repulsive force :

$$-\frac{\partial V_r}{\partial r} = \begin{cases} -r^{-7} & \text{for } r \geq \sqrt{0.2} \\ -0.2^{-3.5} & \text{otherwise} \end{cases}, \tag{3.6}$$

where V_r is the potential resulting from the repulsion between all pairs of atoms (a graphical representation is shown in Figure 3.4).

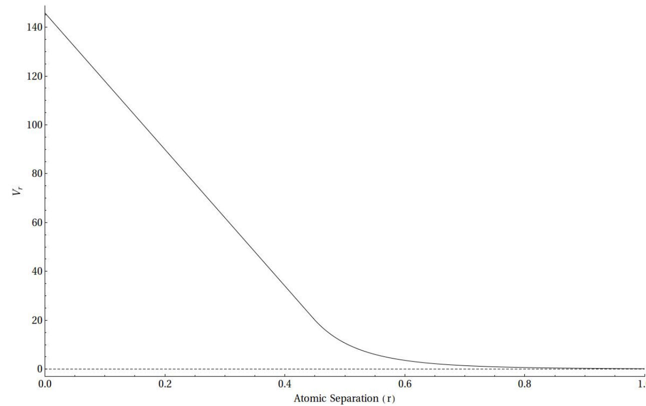


Figure 3.4: Non-Bonded Potential

Within the current study, the 16-atom molecule depicted in Figure 3.5 combined with the energy terms (3.2), (3.4), and (3.6) are used to formulate the energy minimization problem. In the next chapter we describe how PSO can be used to solve this problem.

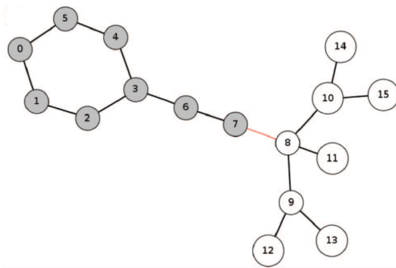


Figure 3.5: Nonlinear Molecular System [4]

CHAPTER 4
IMPLEMENTATION

We begin by defining our search space S. For the problem at hand the search space is simply the atomic configuration space. Since there are sixteen atoms in our molecule and each atom requires three dimensions to describe its position it follows that there are forty eight dimensions in our search space S. Hence, $S = \mathbb{R}^{48}$. Now, for notational purposes define

$$\mathbf{x} = [x_{1,1}, x_{1,2}, x_{1,3}, \ldots, x_{i,j}, \ldots, x_{48,1}, x_{48,2}, x_{48,3}] \in S,$$

where $x_{i,j}$ denotes atom i in dimension j. In terms of PSO, every particle represents an atomic configuration $\mathbf{x} \in S$, and the goal is to find $\mathbf{x}^* \in S$ such that $V(\mathbf{x}^*) \leq V(\mathbf{x})$ for all $\mathbf{x} \in S$. That is, PSO will attempt to find the atomic configuration $\mathbf{x}^*$ that will minimize the potential energy V.

4.1 Serial Implementation

The PSO algorithm was implemented in serial by allocating three 48 x P matrices for particle position, velocity, and pBest, where P is the number of particles used. A good heuristic when deciding on the number of particles to use is that the number should fall some where between d and $10d$, where d is the number of dimensions in the search space. This gives a range between 48 and 480 particles for this particular problem. We decided to use 240 particles in our search. The position of particle i is column i of the particle matrix. Similarly, the velocity and pBest of particle i is column i of the velocity and pBest matrices, respectively. One 48 x 1 array was allocated for gBest and two 240 x 1 arrays for particle and pBest energies (these last two arrays were used to minimize the number of function evals which reduced run times by a factor of three). In each dimension the particle positions and gBest were uniformly distributed between -4 and 4 while the particle velocities were uniformly distributed between -2 and 2. The swarm radius and maximum iterations were

used as termination conditions. The algorithm was stopped if the swarm radius was less than 10^{-9} or the number of iterations had exceeded 20,000. Once the algorithm began its search the velocity matrix was updated using (2.2) above, then using the array functionality of FORTRAN the particle matrix was updated with the loop

```
DO j = 1 , 240
    particle(: , j) = particle(: , j) + velocity(: , j)
END DO
```

No position boundaries were implemented but we did, however, enforce a maximum velocity. We set $V_{max,i} = 2.0$ for $i = 1, 48$. The range of the initializations, number of particles used, V_{max}, and the termination conditions are summarized in Table 4.1. Additional simulations with more particles, larger swarm radii, and different limits on the number of iterations were also performed.

Table 4.1: PSO Settings

# of Particles	**Initializations**			$\mathbf{V}_{max}$	**Termination Conditions**	
	particle	gBest	velocity		Swarm Radius	Max Iterations
240	$U(-4,4)$	$U(-4,4)$	$U(-2,2)$	2.0	10^{-9}	20,000

4.2 Parallel Implementation

The next chapter describes the methods we used to determine a set of good parameters for PSO. Due to the inherent randomness of PSO, we were required to run a large number trials using different sets of parameters as a way to compare performance. A typical run could last any where between 6 and 12 seconds. This quickly became very time consuming, so we set about parallelizing the algorithm using the *Message Passing Interface* (MPI) library. The serial code was parallelized by having the first $k-1$ cores generate $240/k$ columns of the particle, velocity, and pBest matrices, where k is the number of cores running in parallel. The last core generated the remaining $240/k$ + MOD(240,k) columns of the particle, velocity,

and pBest matrices. Each core also allocated a 48 x 1 gBest_local array and a 48 x k matrix called gBest_global. After each time step the MPI command *allgather* was used to collect all the local gBest arrays into the gBest_global matrix. Each core then evaluated the potential energy function using every column of the gBest_global matrix and updated gBest_local accordingly.

CHAPTER 5

RESULTS

The parameters ω, c_1, and c_2 are often problem dependent, and this problem proved to be no exception. The default value $\omega(0) = 1.2$ is widely accepted as a good starting value for the inertial parameter, and this is what we used. The linearly decreasing scheme in equation (2.8) was also employed with $\omega(T) = 0.0$. The algorithm performance was very sensitive to the values set for c_1 and c_2. Two scenarios were encountered when selecting initial values for the cognitive and social parameters:

1. The swarm would never collapse and the algorithm would terminate because the maximum iterations had been exceeded, or

2. The swarm would collapse very quickly before a good solution had been found.

After some preliminary investigations it was found that with $c_1 = 0.3$ and $c_2 = 0.9$ the algorithm would perform relatively well compared to other parameter choices. The algorithm was not converging prematurely and was not exceeding the maximum number of iterations, thereby demonstrating a good balance of exploration and exploitation. However it was obvious that the cognitive and social parameters needed to be fine "tuned".

5.1 Parameter Selection

The base line for these trials were the parameter values $c_1 = 0.3$ and $c_2 = 0.9$. We began by fixing one of the parameters at the base line value and then altering the other parameter by a small amount. Then, the algorithm used these parameters in 10,000 separate trials. The final energies returned by PSO for these trials are displayed in Figure 5.1 and Figure 5.2 and were also ordered into percentiles which are given in Table 5.1.

Table 5.1: Parameter Sweep

c_1	**0.15**	**0.45**	**0.60**	**0.75**
E_{min}	0.2842	0.2842	0.2841	0.2842
E_{90}	0.3424	0.2959	0.2975	0.3086
E_{95}	0.3727	0.3037	0.3060	0.3225
E_{99}	0.5071	0.3390	0.3459	0.3745

(a)

c_2	**0.45**	**1.35**	**1.80**	**2.25**
E_{min}	0.2842	0.2842	0.2842	0.2841
E_{90}	0.3420	0.2993	0.2926	0.2871
E_{95}	0.3897	0.3065	0.2963	0.2890
E_{99}	0.7858	0.3308	0.3105	0.2929

(b)

The tables above are meant to be read from left to right, so for example the entry in the 4th row and 3rd column of the Table 5.1(a) reports that, for parameters $c_1 = 0.60$ and $c_2 = 0.90$, 95% of the trials end with energy values E such that

$$E_{min} = 0.2841 \leq E \leq 0.3060 = E_{95}.$$

Alternatively, we can say that 95% trials returned energies that lie within 0.0219 of the minimum energy found which is $E_{min} = 0.2841$. In fact this interpretation provides us with a good measure of performance; the smaller the difference between a percentile and the minimum energy found implies a smaller deviation from the minimum on average. Let Δ_k be the difference between E_{min} and the kth percentile. The performance measure is presented in Table 5.2 using this notation.

Table 5.2: Performance Measure

c_1	**0.15**	**0.45**	**0.60**	**0.75**
Δ_{90}	0.0582	0.0117	0.0134	0.0244
Δ_{95}	0.0885	0.0195	0.0219	0.0383
Δ_{99}	0.2229	0.0548	0.0618	0.0903

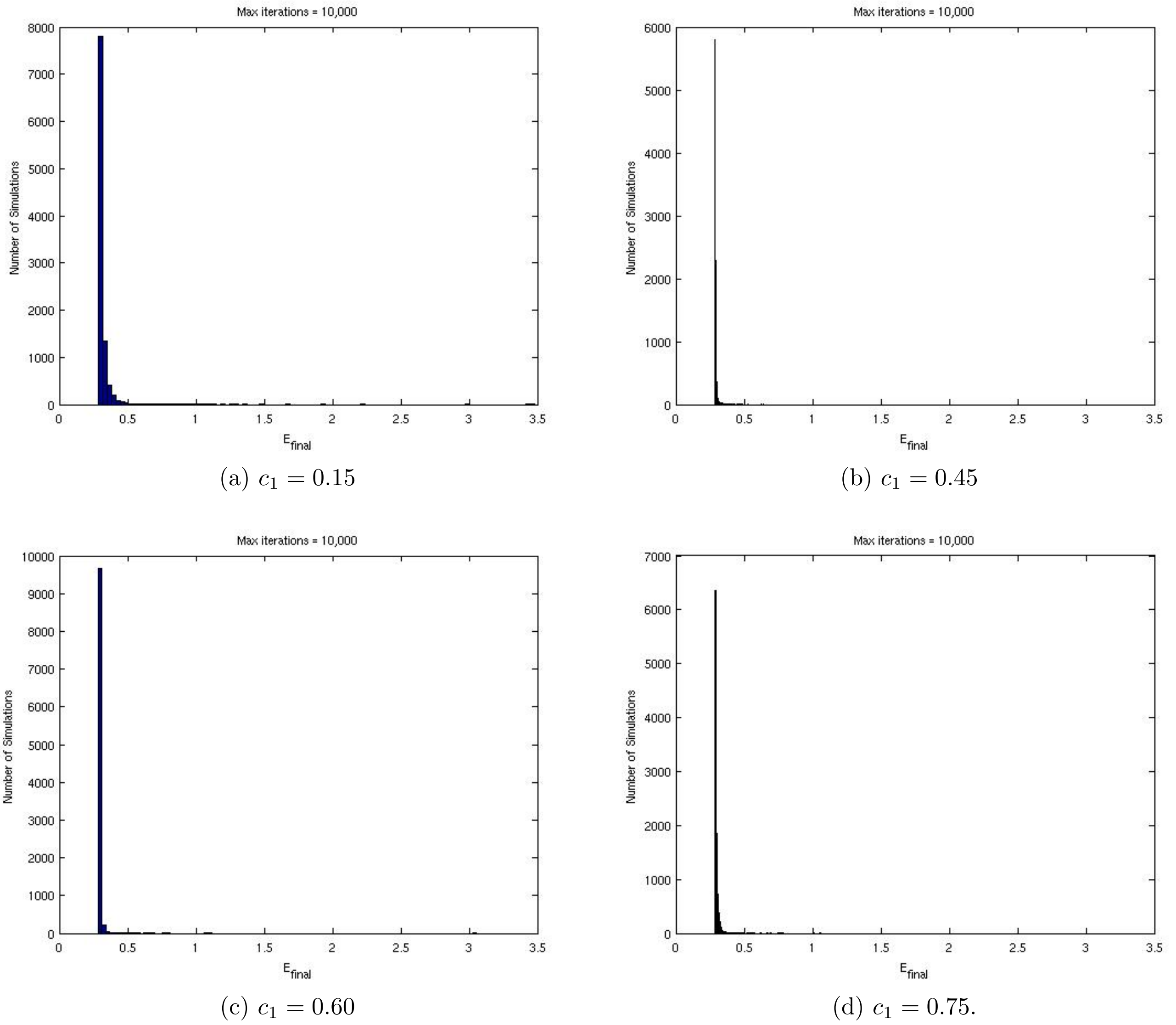
(a)

c_2	**0.45**	**1.35**	**1.80**	**2.25**
Δ_{90}	0.0578	0.0151	0.0084	0.0030
Δ_{95}	0.1055	0.0223	0.0121	0.0049
Δ_{99}	0.5016	0.0466	0.0263	0.0088

(b)

One can clearly see from Table 5.2 that the parameters $c_1 = 0.45$ and $c_2 = 2.25$ out performed all other parameter values.

The histograms shown in Figure 5.1 and Figure 5.2 were generated with 100 bins so that the width of each bin accounts for one percent of the spread in returned energies.



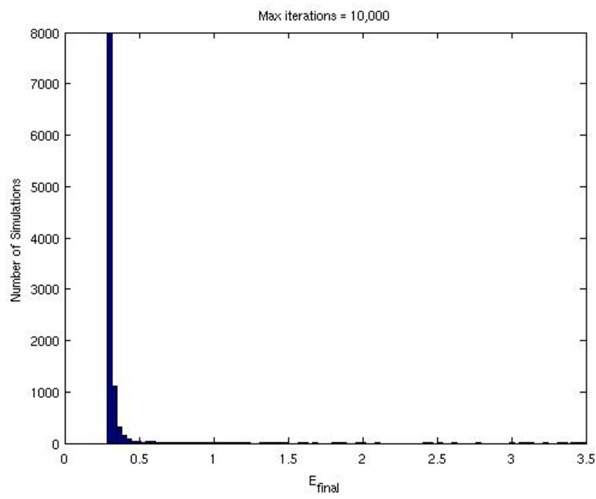
(a) $c_1 = 0.15$

(b) $c_1 = 0.45$

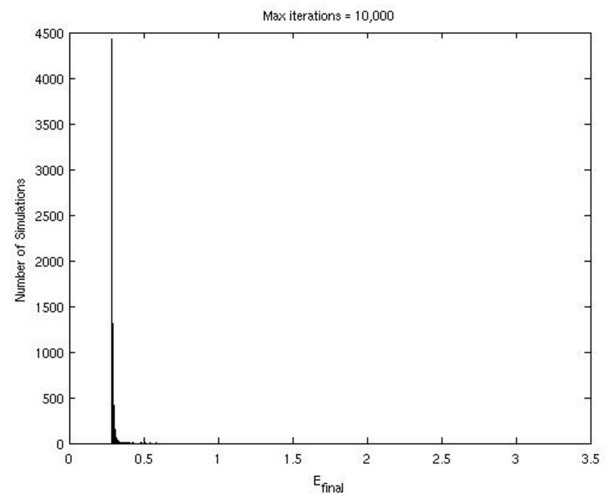
(c) $c_1 = 0.60$

(d) $c_1 = 0.75$.

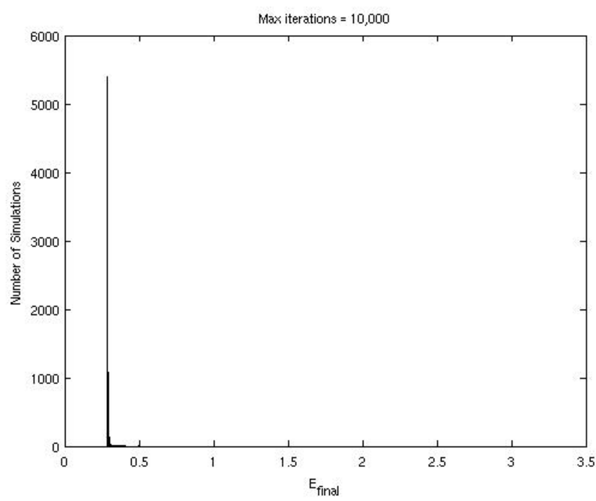
Figure 5.1: Energy Histogram with $c_2 = 0.90$



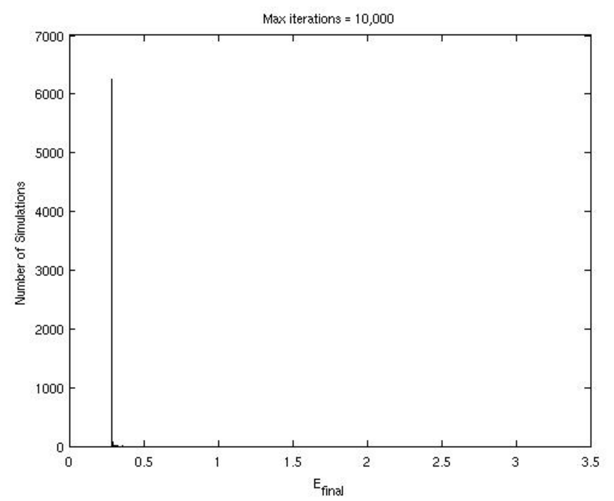
(a) $c_2 = 0.45$



(b) $c_2 = 1.35$



(c) $c_2 = 1.80$



(d) $c_2 = 2.25$

Figure 5.2: Energy Histogram with $c_1 = 0.30$

5.2 Minimal Energy Configuration

In the previous section, PSO reported that the minimal energy of the molecule under investigation was $E_{min} = 0.2841$. Figure 5.3 depicts the 3-dimensional representation of this minimal energy configuration.

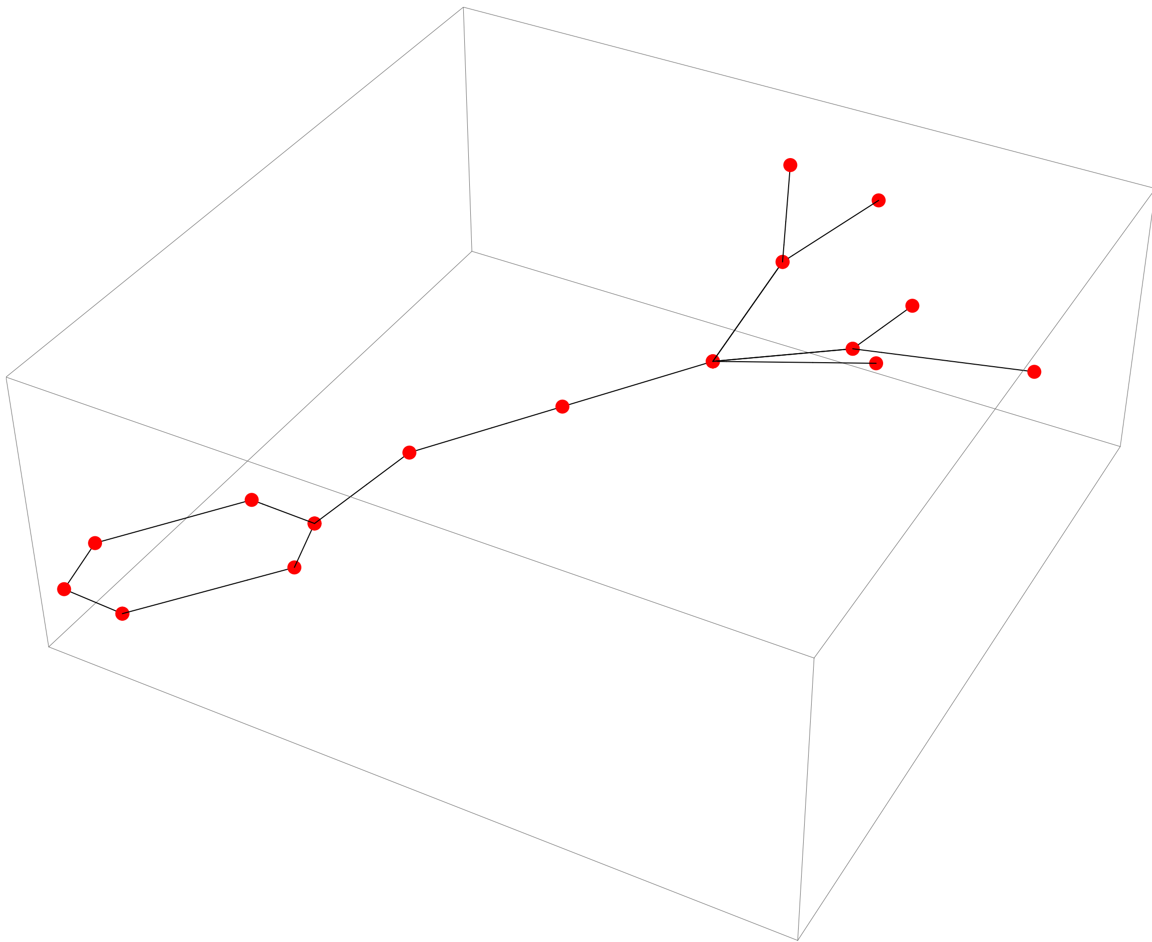


Figure 5.3: Minimal Energy Configuration

5.3 Comparisons

According to the No Free Lunch Theorem (NFLT) [14], there is no single algorithm that will out perform all other algorithms across all optimization problems. Therefore, we compared PSO's performance with several other optimization techniques to gauge its utility. To begin the comparisons we ran PSO with the parameters $c_1 = 0.45$ and $c_2 = 2.25$, along

with the settings in Table 4.1 and recorded the minimum energy, number of function evals, iterations, and run time. For our first comparison we used the so-called "brute force" method where points in the search space were randomly selected and evaluated using the potential energy function V. Each component was uniformly distributed between -4.0 and 4.0 which was the same initialization scheme implemented for PSO in Table 4.1. Exactly 3,107,280 points were generated so that the number of function evaluations performed by both the brute force method and PSO would be the same. The second comparison was done using the conjugate gradient method (CG). CG requires an initial starting point $\mathbf{x}_0$, so we set $\mathbf{x}_0 = \mathbf{g}(0)$, where $\mathbf{g}(0)$ was the initial *gBest* vector generated by PSO. The results of the comparisons are reported in Table 5.3

Table 5.3: Optimization Method Comparison

	Brute Force	Conjugate Gradient	PSO
min E	80.8174	0.2842	0.2842
function evals	3,107,280	34,050	3,107,280
iterations	-	562	12,946
run time	22 sec	102 sec	40 sec

Not surprisingly, the brute force approach was unable to determine a minimizer for the potential energy while PSO and the conjugate gradient method returned the same minimal energy. PSO was significantly faster in terms of run time but required approximately 91 times more function evals than did the conjugate gradient method. Naturally in a much higher dimensional search space, where one could be dealing with several hundred atoms, PSO's larger number of function evals could become computationally prohibitive. Bear in mind though that using less particles would require less function evals, and in general the number of function evaluations could be greater or lesser given PSO's stochastic nature.

Since the conjugate gradient method's ability to find a minimizer depends on the initial starting point $\mathbf{x}_0$, it seemed natural to provide other starting points to observe whether or not the algorithm determine the minimal energy configuration. Therefore, a set of random points was generated and used to initialize the algorithm. In each case the conjugate gradient

method returned the same minimal energy, 0.2842. This seems to suggest the search space has few local minima. To substantiate this claim, consider the Rastrigrin function

$$R(\mathbf{x}) = 10n + \sum_{i=1}^{n}[x_i^2 + 10cos(2\pi x_i)], \tag{5.1}$$

where $x_i \in [-5.12, 5.12]$. The Rastrigrin function has a global minimum $\mathbf{x}^* = \mathbf{0}$ with $R(\mathbf{x}^*) = 0$ but has a large number of surrounding local minima as Figure 5.4 demonstrates.

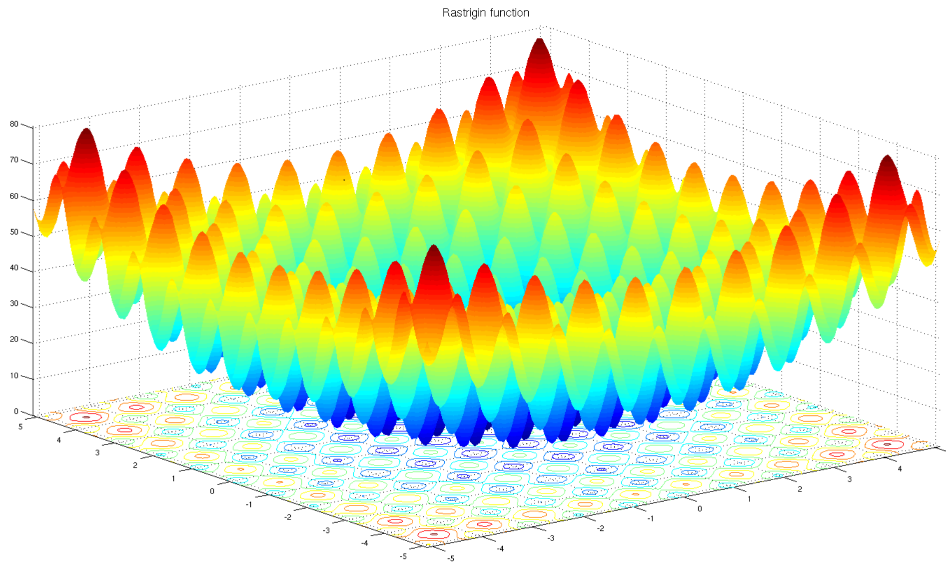


Figure 5.4: Rastrigrin function in 2D

As one might suspect, the conjugate gradient method is only able to find the global minimum if its starting position $\mathbf{x}_0$ is sufficiently close to the origin. For any other starting position the method converges onto a local minimum. Because of this, and the fact that this method was able to find the minimal energy configuration for several randomly chosen starting points, suggests that our search space has little to no local minima.

Another interesting observation is that PSO has no problem locating the global minimum of the Rastrigrin function even in higher dimensions. Using Rastrigrin's function, another comparison was performed between PSO and CG. Again, CG requires an initial starting point $\mathbf{x}_0$, so we set $\mathbf{x}_0 = \mathbf{g}(0)$, where $\mathbf{g}(0)$ was the initial *gBest* vector generated by PSO. Table 5.4 reports the number of dimensions (dim), the minimum function value found (R_{min}), and the

number of function evaluations performed (# evals). Clearly, PSO outperformed CG in this environment, which comes as no surprise since PSO does not use the gradient of a function to find the global minimum. This makes PSO an ideal tool when solving optimization problems in highly multimodal search spaces. This phenomenon of a wide variety of local minima occurs extensively in the field of molecular modeling, as the difference between lesser energy states is often quite small in comparison to the global energy values [15]. Hence, molecular structure can often change from one local energy state to another with minimal variation in the Hamiltonian especially if one considers the inclusion of external forces. Thus the identification of global minimum values over a number of nearby local minima is of great importance.

Table 5.4: Rastrigrin Comparison

	PSO		CG	
dim	R_{min}	# evals	R_{min}	# evals
2	8.1774 x 10^{-9}	1,832	3.9798	56
3	1.7716 x 10^{-6}	11,862	17.3036	42
4	1.8518 x 10^{-10}	12,712	32.8334	90
5	1.5656 x 10^{-6}	5,205	25.8689	105
6	7.1054 x 10^{-14}	93,540	47.7577	144

5.4 Conclusions

This work demonstrates that PSO can be a fast and efficient method for the use of energy minimization of molecular systems. One simply needs to adjust the cognitive and social parameters to the molecular system under investigation, and these adjustments can be achieved by performing a simple parameter sweep like the one that was performed in section 5.1. Furthermore, if the molecular system under investigation has many local minima (stable equilibrium) and one is interested in finding the ground state configuration, then PSO can be the algorithm of choice since it does not use information regarding derivatives of the potential energy function. Finally, for larger molecular systems where computational run time can become an issue, the implementation described in sections 4.1 and 4.2 make the

parallelization of the algorithm a straight forward task.

5.5 Moving Forward

Throughout this work we have demonstrated how one might use PSO to solve an energy minimization problem. Now we present how this approach can be used on a more complex molecules like proteins. Consider globular actin (G-actin) which is a multifunctional protein that is found in all eukaryotic cells and is involved in many important biological processes including cell division, muscle contraction, and organelle movement. G-actin is composed of 375 C^{α} atoms which makes this a very challenging energy minimization problem due to the 1125 dimensional search space. To tackle this problem we would first need to model G-actin with our potential energy function. Hence, we need to know where the chemical bonds are located so that we can specify the entries k_{ij} in the spring constant matrix K. The bond sites for G-actin can be found in the RCSB protein data bank (RCSB PDB) under 1J6Z. With the K matrix in place, we can then specify the entries B_{ijk} in the tensor of angle strength interaction. After this, the potential energy function would be complete and ready for PSO.

REFERENCES CITED

[1] Gerardo Beni and Jing Wang. Swarm intelligence in cellular robotic systems. In *Robots and Biological Systems: Towards a New Bionics?*, pages 703–712. Springer, 1993.

[2] Andries P Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2006.

[3] Yoshikazu Fukuyama and Hirotata Yoshida. A particle swarm optimization for reactive power and voltage control in electric power systems. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 87–93. IEEE, 2001.

[4] Nicholas Guttenberg, James F Dama, Marissa G Saunders, Gregory A Voth, Jonathan Weare, and Aaron R Dinner. Minimizing memory as an objective for coarse-graining. *The Journal of chemical physics*, 138(9):094111, 2013.

[5] J. Kennedy and R. Eberhart. *Particle Swarm Optimization*. Proceedings of the IEEE International Joint Conference on Neural Networks, pages 1942-1948. IEEE Press, 1995.

[6] James Kennedy and Rui Mendes. Population structure and particle swarm performance. 2002.

[7] Majid M Khodier and Christos G Christodoulou. Linear array geometry synthesis with minimum sidelobe level and null control using particle swarm optimization. *Antennas and Propagation, IEEE Transactions on*, 53(8):2674–2679, 2005.

[8] Michael Meissner, Michael Schmuker, and Gisbert Schneider. Optimized particle swarm optimization (opso) and its application to artificial neural network training. *BMC bioinformatics*, 7(1):125, 2006.

[9] Mark M Millonas. Swarms, phase transitions, and collective intelligence. *arXiv preprint adap-org/9306002*, 1993.

[10] ES Peer, F van den Bergh, and AP Engelbrecht. Using neighborhoods with the guaranteed convergence pso, 2003.

[11] Vitorino Ramos, Carlos Fernandes, and Agostinho C Rosa. Social cognitive maps, swarm perception and distributed search on dynamic landscapes. *arXiv preprint nlin/0502057*, 2005.

[12] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34, 1987.

[13] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998.

[14] David H Wolpert and William G Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[15] Robert Zwanzig. *Nonequilibrium statistical mechanics*. Oxford University Press, 2001.

APPENDIX - FORTRAN CODE

Presented below are the codes for the potential function, serial PSO, and the parallelized PSO.

A.1 Potential Function

```
FUNCTION potential(particle,K,B,N)
! Purpose:
! To calculate the potential for the nonlinear molecular model presented
! in the Voth Paper
! Declaration Statement:
INTEGER,PARAMETER :: DBL=KIND(0.0d0)
INTEGER :: i,j,l
REAL(KIND=DBL),DIMENSION(3,N) :: r ! position of each atom
REAL(KIND=DBL) :: d ! atom seperation
REAL(KIND=DBL) :: U_a,U_b,U_r
REAL(KIND=DBL) :: potential
INTEGER,INTENT(IN) :: N ! Number of atoms in the molecule
REAL(KIND=DBL),INTENT(IN),DIMENSION(3*N) :: particle
REAL(KIND=DBL),INTENT(IN),DIMENSION(N,N) :: K
REAL(KIND=DBL),INTENT(IN),DIMENSION(N,N,N) :: B

r = RESHAPE(particle,(/3,N/))

! Calculate U_b
U_b = 0.0_DBL
DO i = 1,N-1
```

```
DO j = i+1,N
IF (K(i,j) /= 0.0) THEN
U_b = SUM((r(:,i)-r(:,j))**2) + U_b
END IF
END DO
END DO
U_b = 0.5_DBL*U_b

! Calculate U_a
U_a = 0.0_DBL
DO i = 1,N-2
DO j = i+1,N-1
DO l = j+1,N
IF(B(i,j,l)/=0.0)THEN
U_a=B(i,j,l)*SUM((r(:,i)-r(:,j))*(r(:,j)-r(:,l)))&
/(sqrt(SUM((r(:,i)-r(:,j))**2))*sqrt(SUM((r(:,j)-r(:,l))**2)))+U_a
END IF
END DO
END DO
END DO
U_a = -U_a

! Calculate U_r
U_r = 0.0_DBL
DO i = 1,N-1
DO j = i+1,N
d = sqrt(SUM((r(:,i)-r(:,j))**2))
```

```
IF(d>sqrt(0.2))THEN

U_r=REAL(1.0/6.0,DBL)*(1/(d**6))+U_r

ELSE

U_r = -(0.2_DBL)**(-7.0/2.0)*d + REAL(7.0/6.0,DBL)*(0.2_DBL)**(-3) + U_r

END IF

END DO

END DO

potential = U_a + U_b + U_r END FUNCTION
```

A.2 Serial PSO

```
PROGRAM PSO_Voth

IMPLICIT NONE

! Declaration Statement

INTEGER,PARAMETER :: DBL=KIND(0.0d0)

REAL(KIND=DBL),ALLOCATABLE,DIMENSION(:,:) :: particle,velocity,pBest

REAL(KIND=DBL),ALLOCATABLE,DIMENSION(:) :: gBest,particle_energy,pBest_energy

REAL(KIND=DBL),ALLOCATABLE,DIMENSION(:,:) :: pos

REAL(KIND=DBL),DIMENSION(16,16) :: K ! Matrix of spring constants

REAL(KIND=DBL),DIMENSION(16,16,16) :: B ! Tensor of angle interaction strength

REAL(KIND=DBL) :: gBest_energy

REAL(KIND=DBL) :: swarm_radius,d1

REAL(KIND=DBL),PARAMETER :: c1 = 0.45 ! cognitive parameter

REAL(KIND=DBL),PARAMETER :: c2 = 2.25 ! social parameter

REAL(KIND=DBL) :: omega_max = 1.147 ! initial inertial term

REAL(KIND=DBL) :: omega_min = 0.0 ! initial inertial term

REAL(KIND=DBL) :: omega ! iteration dependent inertial term

REAL(KIND=DBL) :: r1,r2 ! random numbers

 REAL(KIND=DBL) :: potential ! Obective function
```

```
REAL :: start_time,end_time
INTEGER :: N,P ! number of atoms and particles, respectively
INTEGER :: i,j,l,iostatus ! loop indices INTEGER :: iterations ! counter
! Read in K matrix and B tensor
OPEN(Unit=1,FILE='K_Matrix.txt',STATUS='OLD',ACTION='READ',IOSTAT=iostatus)
OPEN(Unit=2,FILE='B_Tensor.txt',STATUS='OLD',ACTION='READ',IOSTAT=iostatus)
DO i = 1,16
DO j = 1,16
READ(1,*) K(i,j)
DO l = 1,16
READ(2,*) B(i,j,l)
END DO
END DO
END DO
CLOSE(Unit=1) C
CLOSE(UNIT=2)
! Get the number of particles and dimensions
WRITE(*,*)'Enter the number of particles and the number of atoms:'
READ(*,*) P,N
! Allocate memory
ALLOCATE(particle(3*N,P),velocity(3*N,P),pBest(3*N,P),&
gBest(3*N),particle_energy(P),pBest_energy(P),pos(3,N))
! Initialize random seed
CALL INIT_RANDOM_SEED()
! Initialize particle, velocity, and pBest arrays
velocity = 0.0_DBL
DO i = 1,3*N
```

```
DO j = 1,P
CALL RANDOM_NUMBER(r1)
CALL RANDOM_NUMBER(r2)
particle(i,j) = r1*(8.0_DBL)-4.0_DBL
END DO
END DO
pBest = particle ! pBest coincides with the particles position at initialization
! Compute particle and pBest energies
DO i = 1,P
particle_energy(i) = potential(particle(:,i),K,B,N)
END DO
pBest_energy = particle_energy
! Initialize gBest and gBest_energy
DO i = 1,3*N
CALL RANDOM_NUMBER(r1)
gBest(i) = r1*(8.0_DBL)-4.0_DBL
END DO
gBest_energy = potential(gBest,K,B,N)
! Determine gBest
DO i = 1,P
IF (pBest_energy(i) < gBest_energy) THEN
gBest = particle(:,i)
END IF
END DO
! Calculate the swarm radius
swarm_radius = SQRT(SUM((particle(:,1)-gBest)**2))
DO i = 2,P  d1 = SQRT(SUM((particle(:,i)-gBest)**2))
```

```
IF (d1 > swarm_radius) THEN

swarm_radius = d1

END IF

END DO

! Initialize iterations iterations = 0

! Start Timer CALL CPU_TIME(start_time)

DO WHILE (swarm_radius > 1.0E-9.AND. iterations < 10000)

iterations = iterations + 1 ! counter

omega = omega_max-((omega_max-omega_min)*REAL(iterations,DBL))/REAL(10000,DBL)

! Update velocity

DO i = 1,3*N

DO j = 1,P

CALL RANDOM_NUMBER(r1)

CALL RANDOM_NUMBER(r2)

velocity(i,j) = omega*velocity(i,j) + r1*c1*(pBest(i,j) - particle(i,j))&

+ r2*c2*(gBest(i) - particle(i,j))

IF (velocity(i,j) > 4.0_DBL) THEN

velocity(i,j) = 4.0_DBL

ELSE IF (velocity(i,j) < -4.0_DBL) THEN

velocity(i,j) = -4.0_DBL

END IF

END DO

END DO

! Update position

DO i = 1,P

particle(:,i) = particle(:,i) + velocity(:,i)

END DO
```

```
! Update pBest and pBest_energy
DO i = 1,P
particle_energy(i) = potential(particle(:,i),K,B,N)
IF (particle_energy(i) < pBest_energy(i)) THEN
pBest(:,i) = particle(:,i)
pBest_energy(i) = particle_energy(i)
END IF
END DO
! Update gBest
DO i = 1,P
IF (pBest_energy(i) < gBest_energy) THEN
gBest = pBest(:,i)
gBest_energy = pBest_energy(i)
END IF
END DO
! Calculate Swarm Radius
swarm_radius = SQRT(SUM((particle(:,1)-gBest)**2))
DO i = 2,P  d1 = SQRT(SUM((particle(:,i)-gBest)**2))
IF (d1 > swarm_radius) THEN
swarm_radius = d1
END IF
END DO
END DO
! Stop timer CALL CPU_TIME(end_time)
! Write position vector to file
pos=RESHAPE(gbest,(/3,N/))
OPEN(UNIT=1,FILE='X.txt',STATUS='REPLACE',ACTION='WRITE',IOSTAT=iostatus)
```

```
OPEN(UNIT=2,FILE='Y.txt',STATUS='REPLACE',ACTION='WRITE',IOSTAT=iostatus)
OPEN(UNIT=3,FILE='Z.txt',STATUS='REPLACE',ACTION='WRITE',IOSTAT=iostatus)
WRITE(1,*) pos(1,:)
WRITE(2,*) pos(2,:)
WRITE(3,*) pos(3,:)
CLOSE(UNIT=1)
CLOSE(UNIT=2)
CLOSE(UNIT=3)
! Display Results
WRITE(*,*)'Swarm Radius = ',swarm_radius
WRITE(*,*)'Function eval at gBest = ',gBest_energy
WRITE(*,*)'iterations = ',iterations
WRITE(*,*)'T1 =',end_time - start_time
! Deallocate memory DEALLOCATE(particle,velocity,pBest,gBest,pos)
END PROGRAM
```

A.3 Parallel PSO

```
PROGRAM PSO_Voth_parallel
USE MPI
IMPLICIT NONE
! Declaration Statement
INTEGER,PARAMETER :: DBL=KIND(0.0d0)
REAL(KIND=DBL),ALLOCATABLE,DIMENSION(:,:) :: particle,velocity,pBest,gBest_global,pos
REAL(KIND=DBL),ALLOCATABLE,DIMENSION(:) :: gBest_local,particle_energy,pBest_energy,&
gBest_global_energy
REAL(KIND=DBL) :: gBest_local_energy
REAL(KIND=DBL),PARAMETER :: c1 = 1.27 ! cognitive parameter
REAL(KIND=DBL),PARAMETER :: c2 = 1.54 ! social parameter
```

```
REAL(KIND=DBL),PARAMETER :: V_max = 1.0 ! maximum velocity
REAL(KIND=DBL) :: omega_max = 1.147 ! initial inertial term
REAL(KIND=DBL) :: omega_min = 0.0 ! initial inertial term
REAL(KIND=DBL) :: omega ! iteration dependent inertial term
REAL(KIND=DBL) :: r1,r2 ! random numbers
REAL(KIND=DBL) :: potential ! Obective function
REAL(KIND=DBL) :: local_radius, swarm_radius, d ! Swarm radius termination condition
REAL(KIND=DBL),DIMENSION(16,16) :: K ! Matrix of spring constants
REAL(KIND=DBL),DIMENSION(16,16,16) :: B ! Tensor of angle interaction strength
REAL :: start_time, end_time
INTEGER :: iostatus,ierror,my_rank,num_cores ! Mpi variables
INTEGER :: N=16,P=131 ! number of dimensions and particles, respectively
INTEGER :: i,j,l ! loop indices
INTEGER :: iterations ! counter
INTEGER :: max_iter = 20000 ! termination condition
! Initialize MPI
CALL MPI_INIT(ierror)
CALL MPI_Comm_rank(MPI_COMM_WORLD,my_rank,ierror)
CALL MPI_Comm_size(MPI_COMM_WORLD,num_cores,ierror)
! Read in K matrix and B tensor
OPEN(Unit=1,FILE='K_Matrix.txt',STATUS='OLD',ACTION='READ',IOSTAT=iostatus)
OPEN(Unit=2,FILE='B_Tensor.txt',STATUS='OLD',ACTION='READ',IOSTAT=iostatus)
DO i = 1,16
DO j = 1,16
READ(1,*) K(i,j)
DO l = 1,16
READ(2,*) B(i,j,l)
```

```
END DO
END DO
END DO
CLOSE(Unit=1)
CLOSE(UNIT=2)
! Allocate memory
IF (my_rank < num_cores-1) THEN
ALLOCATE(particle(3*N,1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)),&
velocity(3*N,1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)),&
pBest(3*N,1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)),&
gBest_local(3*N),particle_energy(1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)),&
pBest_energy(1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)),&
gBest_global(3*N,num_cores),gBest_global_energy(num_cores))
ELSE
ALLOCATE(particle(3*N,1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)&
+MOD(P,num_cores)),&
velocity(3*N,1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)+MOD(P,num_cores)),&
pBest(3*N,1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)+MOD(P,num_cores)),&
gBest_local(3*N),particle_energy(1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)&
+MOD(P,num_cores)),&
pBest_energy(1+my_rank*(P/num_cores):(my_rank+1)*(P/num_cores)+MOD(P,num_cores)),&
gBest_global(3*N,num_cores),gBest_global_energy(num_cores)) END IF
! Initialize random seed
CALL INIT_RANDOM_SEED(my_rank)
! Start Timer
IF (my_rank==0) THEN
CALL CPU_TIME(start_time)
```

```
END IF
! Initialize particle, velocity, and pBest arrays
velocity = 0.0_DBL
DO i = 1,3*N
IF (my_rank < num_cores-1) THEN
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)
CALL RANDOM_NUMBER(r1)
CALL RANDOM_NUMBER(r2)
particle(i,j) = r1*(4.0_DBL)-2.0_DBL
END DO
ELSE
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)+MOD(P,num_cores)
CALL RANDOM_NUMBER(r1)
CALL RANDOM_NUMBER(r2)
particle(i,j) = r1*(4.0_DBL)-2.0_DBL
END DO
END IF
END DO
pBest = particle ! pBest coincides with the particles position at initialization
! Initialize gBest_local and gBest_local_energy
DO i = 1,3*N
CALL RANDOM_NUMBER(r1)
gBest_local(i) = r1*(4.0_DBL)-2.0_DBL
END DO
gBest_local_energy = potential(gBest_local,K,B,N)
! Determine gBest_local and gBest_local_energy
IF (my_rank < num_cores-1) THEN
```

```
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)
particle_energy(j) = potential(particle(:,j),K,B,N)
pBest_energy(j) = particle_energy(j)
IF (pBest_energy(j) < gBest_local_energy) THEN
gBest_local = particle(:,j)
END IF
END DO
ELSE
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)+MOD(P,num_cores)
particle_energy(j) = potential(particle(:,j),K,B,N)
pBest_energy(j) = particle_energy(j)
IF (pBest_energy(j) < gBest_local_energy) THEN
gBest_local = particle(:,j)
END IF
END DO
END IF
!Gather all gBest_local and gBest_local_energy to determine gBest_global
CALL MPI_ALLGATHER(gBest_local,3*N,MPI_DOUBLE_PRECISION,gBest_global,3*N,
& MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierror)
CALL MPI_ALLGATHER(gBest_local_energy,1,MPI_DOUBLE_PRECISION,gBest_global_energy,1,
& MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierror)
DO j = 1,num_cores
IF (gBest_global_energy(j) < gBest_local_energy) THEN
gBest_local_energy = gBest_global_energy(j)
gBest_local = gBest_global(:,j)
END IF
END DO
```

```
! Calculate Swarm Radius
IF (my_rank < num_cores-1) THEN
local_radius = SQRT(SUM((particle(:,1+my_rank*(P/num_cores))-gBest_local)**2))
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)
d = SQRT(SUM((particle(:,j)-gBest_local)**2))
IF (d > local_radius) THEN
local_radius = d
END IF
END DO
ELSE
local_radius = SQRT(SUM((particle(:,1+my_rank*(P/num_cores))-gBest_local)**2))
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)+MOD(P,num_cores)
d = SQRT(SUM((particle(:,j)-gBest_local)**2))
IF (d > local_radius) THEN
local_radius = d
END IF
END DO
END IF
CALL MPI_ALLREDUCE(local_radius,swarm_radius,1,MPI_DOUBLE_PRECISION,MPI_MAX,
& MPI_COMM_WORLD,ierror)
! Initialize iterations iterations = 0
DO WHILE ((swarm_radius > 1E-09) .AND. (iterations < max_iter))
CALL INIT_RANDOM_SEED(my_rank)
iterations = iterations + 1 ! counter
omega = omega_max-((omega_max-omega_min)*REAL(iterations,DBL))/REAL(max_iter,DBL)
! Update velocity
DO i = 1,3*N
```

```
IF (my_rank < num_cores-1) THEN
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)
CALL RANDOM_NUMBER(r1)
CALL RANDOM_NUMBER(r2)
velocity(i,j) = omega*velocity(i,j) + r1*c1*(pBest(i,j) - particle(i,j))&
+ r2*c2*(gBest_local(i) - particle(i,j))
! Apply velocity boundaries
IF (velocity(i,j) > V_max) THEN
velocity(i,j) = V_max
ELSE IF (velocity(i,j) < -V_max) THEN
velocity(i,j) = -V_max
END IF
END DO
ELSE
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)+MOD(P,num_cores)
CALL RANDOM_NUMBER(r1)
CALL RANDOM_NUMBER(r2)
velocity(i,j) = omega*velocity(i,j) + r1*c1*(pBest(i,j) - particle(i,j))&
+ r2*c2*(gBest_local(i) - particle(i,j))
! Apply velocity boundaries
IF (velocity(i,j) > V_max) THEN
velocity(i,j) = V_max
ELSE IF (velocity(i,j) < -V_max) THEN
velocity(i,j) = -V_max
END IF
END DO
END IF
```

```
END DO
! Update position
IF (my_rank < num_cores-1) THEN
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)
particle(:,j) = particle(:,j) + velocity(:,j)
END DO
ELSE
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)+MOD(P,num_cores)
particle(:,j) = particle(:,j) + velocity(:,j)
END DO
END IF
! Update pBest
IF (my_rank < num_cores-1) THEN
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)
particle_energy(j) = potential(particle(:,j),K,B,N)
IF (particle_energy(j) < pBest_energy(j)) THEN
pBest(:,j) = particle(:,j)
pBest_energy(j) = particle_energy(j)
END IF
END DO
ELSE
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)+MOD(P,num_cores)
particle_energy(j) = potential(particle(:,j),K,B,N)
IF (particle_energy(j) < pBest_energy(j)) THEN
pBest(:,j) = particle(:,j)
pBest_energy(j) = particle_energy(j)
END IF
```

```
END DO
END IF
! Update gBest_local, gBest_local_energy, and gBest_global
IF (my_rank < num_cores-1) THEN
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)
IF (pBest_energy(j) < gBest_local_energy) THEN
gBest_local = pBest(:,j)
gBest_local_energy = pBest_energy(j)
END IF
END DO
ELSE
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)+MOD(P,num_cores)
IF (pBest_energy(j) < gBest_local_energy) THEN
gBest_local = pBest(:,j)
gBest_local_energy = pBest_energy(j)
END IF
END DO
END IF
CALL MPI_ALLGATHER(gBest_local,3*N,MPI_DOUBLE_PRECISION,gBest_global,3*N,
&  MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierror)
CALL MPI_ALLGATHER(gBest_local_energy,1,MPI_DOUBLE_PRECISION,gBest_global_energy,1,
&  MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierror)
DO j = 1,num_cores
IF (gBest_global_energy(j) < gBest_local_energy) THEN
gBest_local_energy = gBest_global_energy(j)
gBest_local = gBest_global(:,j)
END IF
```

```
END DO
! Calculate Swarm Radius
IF (my_rank < num_cores-1) THEN
local_radius = SQRT(SUM((particle(:,1+my_rank*(P/num_cores))-gBest_local)**2))
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)
d = SQRT(SUM((particle(:,j)-gBest_local)**2))
IF (d > local_radius) THEN
local_radius = d
END IF
END DO
ELSE
local_radius = SQRT(SUM((particle(:,1+my_rank*(P/num_cores))-gBest_local)**2))
DO j = 1+my_rank*(P/num_cores),(my_rank+1)*(P/num_cores)+MOD(P,num_cores)
d = SQRT(SUM((particle(:,j)-gBest_local)**2))
IF (d > local_radius) THEN
local_radius = d
END IF
END DO
END IF
CALL MPI_ALLREDUCE(local_radius,swarm_radius,1,MPI_DOUBLE_PRECISION,MPI_MAX,
&  MPI_COMM_WORLD,ierror)
END DO
IF (my_rank==0) THEN
CALL CPU_TIME(end_time)
WRITE(*,*)'T = ',end_time-start_time
WRITE(*,*)'iterations = ',iterations
WRITE(*,*)'Final Energy = ',gBest_local_energy
```

```
WRITE(*,*)'Swarm_Radius = ',swarm_radius

END IF

! Deallocate memory

DEALLOCATE(particle,velocity,pBest,gBest_local,gBest_global,& particle_energy,&

pBest_energy,gBest_global_energy)

CALL MPI_FINALIZE(ierror)

END PROGRAM
```