

2002

A Dynamical Perspective on Enzymatic Catalysis

Shiying Shang

College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>

 Part of the [Biochemistry Commons](#)

Recommended Citation

Shang, Shiying, "A Dynamical Perspective on Enzymatic Catalysis" (2002). *Dissertations, Theses, and Masters Projects*. William & Mary. Paper 1539626362.

<https://dx.doi.org/doi:10.21220/s2-03fr-f764>

This Thesis is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

A Dynamical Perspective on Enzymatic Catalysis

A Thesis

Presented to

The Faculty of the Department of Chemistry

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Master of Arts

by

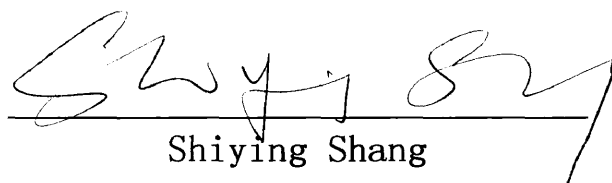
Shiying Shang

2002

APPROVAL SHEET

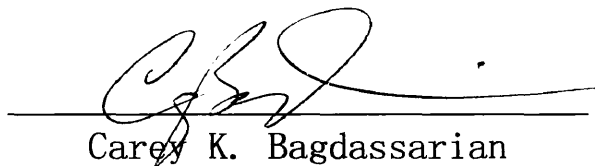
This Thesis is submitted in partial fulfillment of the
requirements for the degree of

Master of Arts

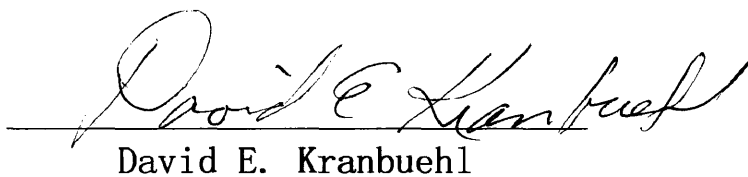


Shiyong Shang

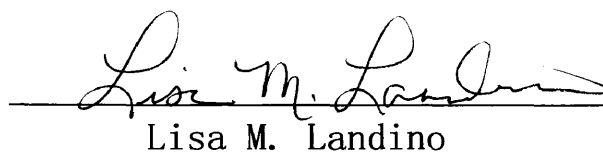
Approved, May 2002



Carey K. Bagdassarian



David E. Kranbuehl



Lisa M. Landino

Table of Contents

Table of Contents.....	iii
Acknowledgments.....	vi
List of Tables.....	viii
List of Figures.....	ix
List of Abbreviations and Symbols.....	x
Abstract.....	xiii
1. General Introduction.....	2
1.1 Purpose.....	2
1.2 A General Picture and Previous Work.....	3
1.3 Model and Molecular Dynamics Simulations.....	6
1.4 Genetic Algorithms.....	10
1.5 Long-Term Goals.....	11

Part I

2. Molecular Dynamics Simulations.....	13
2.1 Introduction to Molecular Dynamics Simulations..	13
2.2 Statistical Mechanics.....	14
2.2.1 Time Average.....	16

2.2.2	Fluctuations.....	18
2.2.3	Correlation Functions.....	19
2.3	Molecular Dynamics Simulations.....	21
2.3.1	Newton' s Equations of Motion.....	23
2.3.2	Integration Algorithms.....	24
	• Verlet Integrator.....	25
	• Verlet Leapfrog Integrator.....	27
	• Velocity Verlet Integrator.....	29
2.3.3	Boundary Conditions.....	31
2.3.4	Temperature Scaling.....	32
2.3.5	Initial Conditions.....	34
2.4	Results and Discussion.....	36
2.5	Conclusion.....	49

Part II

3.	Genetic Algorithms.....	51
----	-------------------------	----

Part III

4.	Appendices.....	54
----	-----------------	----

• How to Use?.....	55
• verlet.h.....	56
• main.cc.....	60
• initialize.cc.....	69
• scale.cc.....	79
Bibliography.....	88
VITA.....	94

Acknowledgments

First and foremost, I would like to thank my advisor, Carey Bagdassarian. He gave me the opportunity to pursue exciting research in his group at the College of William and Mary. His support, encouragement, and guidance were extremely valuable for the development of this work.

I would also like to thank our collaborators, Prof. David Krenbuehl and his student Aftab Hossain. Their insight has been crucial in achieving progress with the computational work reported here. They very much contributed to our discussions regarding the research. These discussions played a key role in building my confidence to solve the research problem, which is a central part of my thesis.

Dr. David Kranbuehl and Dr. Lisa Landino went through large parts of this thesis and gave me useful remarks. I am grateful for their help. I also would like to thank Blair Williams, Ricky Anderson, Alper Kutay for all of their help in the lab, and thanks to all my friends for their support.

Finally, I cannot finish this section without expressing my feelings for my husband and my dearest parents. I am very grateful to their support, which is fundamental in my achievements.

List of Tables

Table 1 - A.....	42
Table 1 - B.....	42
Table 2.....	42
Table 3.....	43
Table 4.....	44

List of Figures

Figure 1.....	7
Figure 2.....	26
Figure 3.....	27
Figure 4.....	30
Figure 5.....	36
Figure 6.....	38
Figure 7.....	39
Figure 8.....	40
Figure 9.....	41
Figure 10.....	45
Figure 11.....	47
Figure 12.....	52
Figure 13.....	53

List of Abbreviations and Symbols

In this list I provide the abbreviations and symbols used throughout the text:

Algorithms

- MD: Molecular Dynamics
- GA: Genetic Algorithm
- GNM: Gaussian Network Model

Abbreviations

- RMS: root mean square
- PE: potential energy
- KE: kinetic energy
- k_{big} : large force constant
- k_{small} : small force constant
- k_f : friction force constant
- C_{cs} : correlation factor of catalytic amino acid (C) and substrate (S).
- C : catalytic amino acid

- S : substrate
- $rmsx$: root mean square deviation in x direction
- $rmsy$: root mean square deviation in y direction
- $EqRx$: equilibrium position in x direction
- $EqRy$: equilibrium position in y direction
- T_d : desired temperature
- T_a : actual temperature

Symbols

- V : potential energy
- K : kinetic energy
- r : position
- v : velocity
- a : acceleration
- M : the number of time steps
- N : the number of atoms
- A : dynamical variable
- F : force
- m : mass

- N_f : degrees of freedom
- k_B : Boltzmann' s constant
- $\rho(\mathbf{v}_{ix})$: probability density for velocity v_{ix}
- ξ : uniform random variable
- ζ : Gaussian-distributed random number
- β : reciprocal temperature $\left(\frac{1}{k_B T} \right)$

ABSTRACT

A Dynamical Perspective on Enzymatic Catalysis

Shiying Shang

This thesis consists of two major parts. In the first part, molecular dynamic simulations are applied to study enzymatic conformational fluctuations. We define a simplified lattice model with each lattice site occupied by an amino acid. This lattice consists of both stiff and loose interactions between neighboring sites. Two neighboring lattice sites are identified as the substrate and catalytic amino acid. Then we integrate Newton's equations of motion for all sites in the system. Lastly, we calculate the correlation function for fluctuations of the substrate and catalytic amino acid, which defines the catalytic efficiency of this enzyme. In the second part, we use Genetic Algorithms and intersect them with molecular dynamics. This procedure will drive the evolution of stiff and flexible interactions between enzymatic domains such that the enzyme's fluctuation dynamics are maximally efficient.

A Dynamical Perspective on Enzymatic Catalysis

Chapter 1

General Introduction

- Purpose

The development of computational methods for biophysical chemistry calculations is a complex and active research area. Much work is directed towards understanding how enzymes are so much smarter than theorists, who cannot even reliably predict what the final folded states of proteins will be. Our research focuses on developing computational tools to address the role of dynamic fluctuation patterns in maximizing the rate of enzymatic catalysis. We propose that an enzyme has evolved for catalysis on two levels. The first level is a static level for substrate recognition and transition-state stabilization. The second level is a dynamic level of complex conformational fluctuation patterns. Our goal will be accomplished via the applications of molecular dynamics simulations and genetic algorithms.

- **A General Picture and Previous Work**

More than 50 years ago, Pauling introduced transition state stabilization theory to explain enzyme functions (Pauling 1948). He hypothesized that enzymes stabilize the transition state by lowering the activation energy thereby maximizing the rate of catalysis. More recently, a ground state destabilization theory whereby enzyme molecules destabilize the substrate rather than stabilize the transition state was proposed by Jencks (Jencks 1975).

Both of these conclusions are based on a static analysis of enzymes. But a real enzyme molecule is dynamic, flexible, and fluctuating. We propose that large-scale complex fluctuation patterns affect the catalytic efficiency to a large extent. A theoretical review of the role of conformational and dynamical changes in ligand binding by Karplus and Petsko showed that molecular dynamics simulations would help understand and predict such biological phenomena (Karplus and Petsko 1990). This conformational flexibility induced by ligand binding has been examined in several experimental studies (Wang et al. 1998;

Greenwald et al. 1999; Hadfield and Mulholland 1999; Loh 1999; Kohen et al. 1999; Zajicek et al. 2000).

Recent literature has demonstrated that small scale dynamic motions have a direct role in catalysis: The coupling between substrate and catalyst may be controlled by amino acid residues far from the active site (Cameron and Benkovic 1997; Miller and Benkovic 1998; and Balabin and Onuchic 2000): Alternate conformations of the enzyme lead to completely different motions, which are essential for catalysis.

Computational evidence and experimental results suggest that fluctuation patterns strongly depend on correlated motions. Radkiewicz and Brooks demonstrate, using molecular dynamics simulations, that strongly-coupled motions of dihydrofolate reductase are linked to catalytic ability (Radkiewicz and Brooks 2000). Ota and Agard, also using MD simulations, revealed that the substrate specificity of an α - lytic protease is dynamically-controlled (Ota and Agard 2001). Gaussian network models (GNM) of tryptophan synthase show that structural

elements control the cooperative transmission of conformational motions (Bahar and Jerignan 1999).

We believe that we can design highly efficient catalysis by optimizing enzymatic fluctuation patterns that develop right after substrate binding. We first model motions of enzymatic amino acids on a computer through molecular dynamics algorithms. Then we apply a genetic algorithm to drive this enzyme to its maximal efficiency.

- **Model and Molecular Dynamics Simulations**

In this work we employ a two-dimensional lattice model of an enzyme, which consists of both conformationally stiff and loose regions. This kind of model is widely used to understand protein-folding problems (Klimov and Thirumalai 1998; Hoang and Cieplak 1998; Socci et al. 1999; Wang et al. 2000; and Williams et al. 2001). Then a molecular dynamics (MD) simulation is applied to this model. The problem with MD is that the scope of the simulations is strictly limited by time. These simulations, with current computational power, typically can run for about a nanosecond (10^{-9} second) for an enzymatic system. But chemical reactions are usually on the time-scale of a millisecond. This problem hinders applicability of this method even at the level of moderately sized systems. However, we simplify an enzyme to its “toy” structure and ignore molecular details of the Van der Waals, electrostatic, torsion, etc. interactions in our small system. Thus our simulations can run for much longer times.

Our model consists of 8×8 amino acids on a square lattice, with each amino acid located at one lattice site. We

define one of these lattice sites, which is located at the center of the matrix, as the substrate (S), and its neighbor as the catalytic amino acid (C).

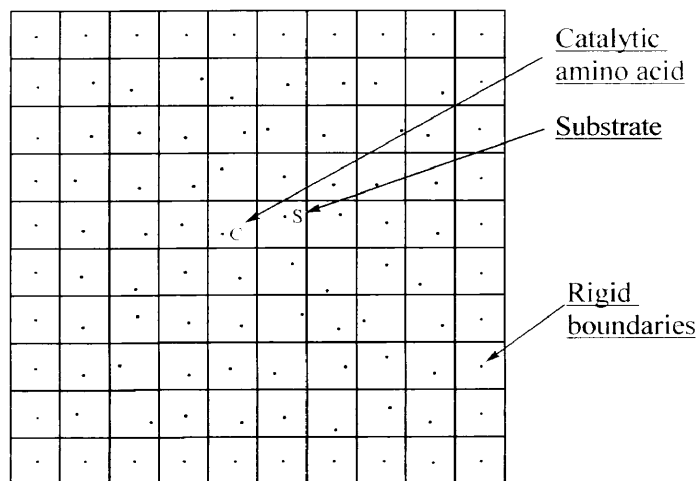


Figure 1. 8×8 model of MD simulations

Then two force constants, k_{big} and k_{small} , are defined to represent stiff and loose regions, respectively. Now the model is ready for MD simulations.

The MD algorithm simulates the motions of a system of particles. This method integrates Newton's equations of motion to advance the atomic motion. This numeric integration is typically done by using Finite Difference methods, such as the Verlet integrator. The central part of this thesis reports the development of the molecular dynamics simulations. The

methodology involved is reported in detail in the first part of the thesis.

Atoms are initially assigned random positions, Gaussian distributed velocities, and randomly distributed force constants for a desired temperature, and a simulation is performed for a few nanoseconds. Strong harmonic interactions prevent each atom from straying too far from its equilibrium position, which leads to protein unfolding. Now, we want to determine the right distribution of small and large force constants that will determine the catalytic efficiency.

During the MD simulations, the more numerous the enzyme-substrate encounters, the higher the probability of the chemical event. The enzyme-substrate encounters, in our simulations can be measured by a correlation factor:

$$C_{cs} = \frac{\langle \Delta \mathbf{x}_c \cdot \Delta \mathbf{x}_s \rangle}{\sqrt{\langle \Delta \mathbf{x}_c^2 \rangle \langle \Delta \mathbf{x}_s^2 \rangle}}$$

where $\Delta \mathbf{x}_c$ and $\Delta \mathbf{x}_s$ represent the deviations of the catalytic residue and the substrate from their equilibrium positions. The brackets indicate averages over equilibrated trajectories. We

want to get a correlation factor that is as negative as possible. That means the catalytic residue and substrate are always moving to the opposite directions, which leads to efficient chemical events.

- **Genetic Algorithms**

To evolve the enzyme to maximal efficiency, a genetic algorithm (GA), based on Natural (Darwinian) Selection, is applied. Genetic algorithms have been widely studied with applications in many fields in the engineering world. They were introduced as a computational analogy of adaptive systems. These methods are modeled loosely on the principles of evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation inducing operators such as mutation and crossover of force constants. After each cycle of GA, the best distributions of force constants are chosen to generate new conformations until the most efficient enzyme fluctuation pattern is found. Genetic algorithms are very helpful and efficient when the search space is large, complex, or poorly understood.

- **Long-Term Goals**

By combining molecular dynamics and genetic algorithms, we are able to optimize the structure of stiff and loose regions in the body of the enzyme. The work will be a novel synthesis of ideas and techniques derived from chemistry, physics, biology and the computer sciences.

Our simulations can also address some biological phenomena such as why the respective enzymes of a thermophilic and a thermophobic organism amazingly perform with the same efficiency at two very different temperatures. The answer is that the enzymatic conformational fluctuation patterns are the same at the respective optimal temperatures.

Since MD simulations can be run at different temperatures, we are able to take advantage of this to establish and compare relations between different enzymes in different thermal conditions.

- **Note**

The reader may find the list of abbreviations and symbols included above helpful.

Part I

Molecular Dynamics

Chapter 2

Molecular Dynamics Simulations

2.1 Introduction to Molecular Dynamics Simulations

Molecular dynamics simulation is a powerful method for exploring the dynamic properties of many - particle systems. The method was originally devised in the 1950s by Alder and Wainwright (Alder and Wainwright 1957, 1959). The essence is simple: calculate the forces acting on the atoms in a molecular system and analyze their motion. MD simulations can provide detailed information on the fluctuations and conformational changes of proteins and nucleic acids. It is a valuable bridge between experiment and theory. Section II of this chapter will focus on the discussion of statistical mechanics. In section III, the integration algorithm, temperature scaling, and some related topics will be presented. Section IV will discuss some practical aspects of our molecular dynamics simulation and some basic analysis.

2.2 Statistical Mechanics

In a molecular dynamics simulation, statistical mechanics connects microscopic simulations and macroscopic properties through mathematical expressions. It helps us to deduce the bulk properties, as opposed to individual atomic properties, of the material (e.g. temperature, potential energy, kinetic energy, etc.). In this section, we provide an overview of three important concepts --- time average, fluctuations, and correlation functions.

- **Time Average:**

A time average is an average taken over a large number of configurations of the system. Features of all individual configurations are supposed to be scattered about this mean value. It helps us to properly calculate the averages.

- **Fluctuations:**

Fluctuations describe how much properties of individual configurations can vary from the average values. Fluctuations,

with no doubts, are very important to all physical processes in the whole system.

- **Correlation Functions:**

Correlation functions measure how fluctuations in the catalytic amino acid residue are correlated with those of the substrate. It reveals the chemical efficiency of the dynamical processes that occur.

2.2.1 Time Average

The time average of some configurational property A is given by

$$\langle A \rangle_{time} = \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \int_{\tau=0}^{\tau} A d\tau \approx \frac{1}{M} \sum_{t=1}^M A$$

where M stands for the number of time steps, τ is the simulation time.

$\tau \rightarrow \infty$ means this measurement is performed over a essentially infinite time duration.

To be more specific:

Average position: $A = R$

$$R(x, y)_i = \langle R(x, y) \rangle = \frac{1}{M} \sum_{j=1}^M \{R(x, y)_i\}_j$$

Average potential energy: $A = V$

$$V = \langle V \rangle = \frac{1}{M} \sum_{j=1}^M \left\{ \sum_{i=1}^N V_i \right\}_j$$

Average kinetic energy: $A = K$

$$K = \langle K \rangle = \frac{1}{M} \sum_{j=1}^M \left\{ \sum_{i=1}^N \frac{1}{2} m_i v_i^2 \right\}_j$$

where M (j): the number of time steps

N (i): the number of atoms

$R(x, y)_i$: the position of atom i

V : potential energy

K : kinetic energy

m_i : the mass of atom i

v_i : the velocity of atom i

2.2.2 Fluctuations

- Fluctuations in potential energy:

Fluctuations in potential energy in the system provide information such as when the system is equilibrated and when we should start sampling for calculations.

- Fluctuations in position:

Calculation of the fluctuations in position helps us to prevent the atoms of interest from straying too far from their equilibrium sites. This information can be obtained from root mean square (RMS) deviation calculations: $R(x) = \sqrt{\langle x^2 \rangle}$

i. e. ,

$$rmsx(x, y) = \sqrt{\frac{\sum_{j=1}^M [R_x(x, y) - EqR_x(x, y)]^2}{M}}$$

$$rmsy(x, y) = \sqrt{\frac{\sum_{j=1}^M [R_y(x, y) - EqR_y(x, y)]^2}{M}}$$

where EqR_x and EqR_y mean the equilibrium positions of R_x and R_y

2.2.3 Correlation Functions

Correlated motions between the substrate and catalytic amino acid are measured by the correlation factor. It reveals how the positions of substrate X_s may be related to the positions of the catalytic amino acid X_c . The explicit form of the correlation function is

$$C_{cs} = \frac{\sum_{i=1}^M [(X_s - \overline{X_s})(X_c - \overline{X_c})]_i}{\sqrt{\sum_{i=1}^M (X_s - \overline{X_s})^2_i} \sqrt{\sum_{i=1}^M (X_c - \overline{X_c})^2_i}}$$

where C_{cs} : the correlation factor of the catalytic amino acid and substrate

X_s : position of substrate

X_c : position of catalytic amino acid

M : configurations (trajectories)

The correlation factor varies in the range $-1 \leq C_{cs} \leq +1$. If this factor is positive, our substrate and catalytic amino acid move in the same direction under most circumstances. On the contrary, an opposite-direction motion, which is in favor of successful chemical events, is related to a negative value of C_{cs} . Therefore, this correlation factor

can be used to measure the probability of collision (i.e., the probability of the chemical event). The more negative the correlation factor, the higher the chance of successful enzyme-substrate encounters.

$$\left\{ \begin{array}{ll} 0 < C_{cs} \leq 1 & \textit{correlated} \\ -1 \leq C_{cs} < 0 & \textit{anti-correlated} \\ C_{cs} = 0 & \textit{uncorrelated} \end{array} \right.$$

2.3 Molecular Dynamics Simulations

Molecular dynamics simulations provide detailed information on the conformational fluctuations of the enzyme. They are based on Newton's equations of motion. We used velocity Verlet as our integration algorithm. In this section, we discuss integration algorithms, boundary conditions, temperature scaling, and initial conditions of our model.

- **Newton's Equations of Motion:**

These help us to calculate the increments in positions, velocities, and acceleration of the whole system from the initial conditions a time step earlier via the laws of classical mechanics.

- **Integration Algorithms:**

Numeric integration of Newton's equations of motion is typically done by using finite difference methods. Here, we will introduce the most common and most basic integration algorithm – the Verlet integrator.

- **Boundary Conditions**

Since periodic boundaries are hard to deal with and are more useful for solid states of matter, we decided to use rigid boundary conditions in our simulation.

- **Temperature Scaling**

To keep the desired temperature of system, we change the current actual temperature Ta to the desired temperature Td by scaling all the velocities by the factor $\sqrt{\frac{Td}{Ta}}$.

- **Initial Conditions**

We chose the size of the time - step by testing the stability of the numerical integrations. To save computation time, our initial velocities are a set of Gaussian-distributed numbers scaled to the desired temperature.

2.3.1 Newton' s Equations of Motion

The motion of an individual atom is usually approximated by Newton' s equations of motion:

$$F = ma \quad \textcircled{1}$$

where F : is the total force on a particle

m : is the mass

a : is the acceleration

Here, F can be given as the derivative of potential energy V , and a can be given as the second derivative of position r .

i. e.,
$$F = -\frac{dV}{dr} \quad \textcircled{2}$$

$$a = \frac{d^2r}{dt^2} \quad \textcircled{3}$$

Combining equations $\textcircled{1}$, $\textcircled{2}$, $\textcircled{3}$, we have

$$-\frac{dV}{dr} = m \frac{d^2r}{dt^2}$$

There is one very important property of Newton' s equations of motion: conservation of energy. Since we fix the temperature to its desired value, the kinetic energy is stable. Consequently, the conservation of potential energy is used to test whether the molecular dynamics simulation is stable and reliable.

2.3.2 Integration Algorithms

A finite difference technique is the standard method for solving an ordinary differential equation such as $-\frac{dV}{dr} = m \frac{d^2 r}{dt^2}$. The general idea is: given the initial positions, velocities, and other dynamic information at time t , we can calculate the positions and velocities at time $t + dt$. At the new positions, the procedure is repeated and another step is made.

Many numerical algorithms have been developed for integrating Newton's equations of motion. The most basic and most common group of integration algorithms is the class of Verlet algorithms, such as the Verlet integrator, Verlet Leapfrog integrator, and Velocity Verlet integrator. Our molecular dynamics program is based on the Velocity Verlet algorithm.

Verlet Integrator

The Verlet integrator is the simplest integration algorithm.

This method is based on a Taylor expansion about $r(t)$:

$$r(t + dt) = r(t) + dt \cdot v(t) + \frac{1}{2} dt^2 a(t) + \dots$$

$$r(t - dt) = r(t) - dt \cdot v(t) + \frac{1}{2} dt^2 a(t) - \dots$$

Then, we add these two equations to give the equation for calculating the positions:

$$r(t + dt) = 2r(t) - r(t - dt) + dt^2 \cdot a(t)$$

The velocities can be obtained from the equation:

$$v(t) = \frac{r(t + dt) - r(t - dt)}{2dt}$$

Using these equations, the new positions and velocities are ready for the next time step.

This can be illustrate in the following figure:

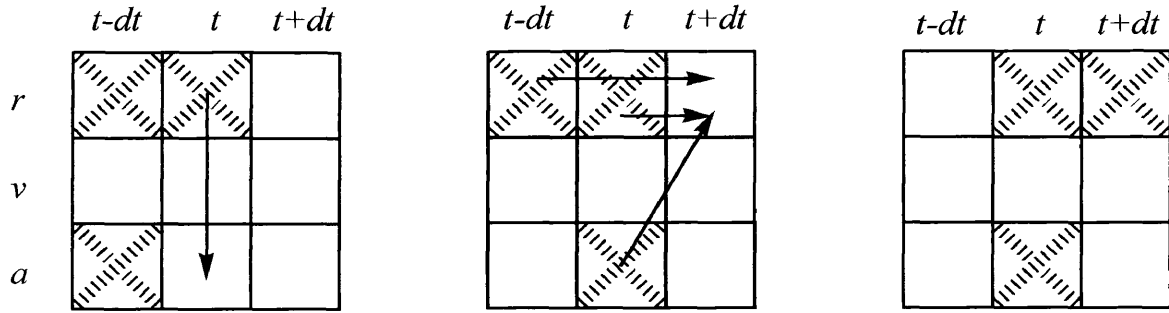


Figure 2. Verlet Integrator

Figure 2: The figure shows successive steps in the implementation of the algorithm. The stored variables are in crossed boxes. The simulations follow the arrows to calculate the advance of features (r , v , or a). The third graph shows the variables at time $t + dt$, while the first graph is for time t . (from M. P. Allen and D. J. Tildesley, 1987. Computer Simulation of Liquids)

Verlet Leapfrog Integrator

The Verlet Leapfrog integrator is an improved algorithm based on Verlet algorithm. The definition of this algorithm is as follows:

$$v\left(t + \frac{dt}{2}\right) = v\left(t - \frac{dt}{2}\right) + a(t) \cdot dt$$

$$r(t + dt) = r(t) + v\left(t + \frac{dt}{2}\right) \cdot dt.$$

The velocities can be calculated from:

$$v(t) = \frac{v\left(t + \frac{dt}{2}\right) + v\left(t - \frac{dt}{2}\right)}{2}.$$

The process is shown as follows,

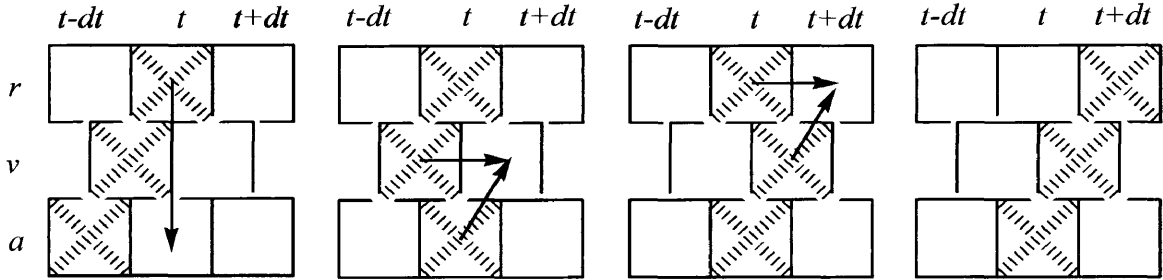


Figure 3. Verlet Leapfrog Intergrator

(from M. P. Allen and D. J. Tildesley, 1987. Computer Simulation of Liquids)

This algorithm is a little more computationally expensive than the Verlet algorithm, but it needs less storage space and improves the evaluation of velocities.

Velocity Verlet Integrator

The Velocity Verlet integrator, the most improved integrator, is used in our simulation. This algorithm stores positions, velocities, and accelerations at the same instant of time. The basic steps are as follows:

$$r(t+dt) = r(t) + dt \cdot v(t) + \frac{1}{2} dt^2 \cdot a(t) \quad \textcircled{1}$$

$$v(t+dt) = v(t) + \frac{1}{2} dt \cdot [a(t) + a(t+dt)] \quad \textcircled{2}$$

The algorithm involves the following steps:

- We start with $r(t)$ and $v(t)$ and calculate the acceleration at this time $a(t)$.
- Then we repeat a loop:
 1. Calculate $r(t+dt)$ by using equation $\textcircled{1}$
 2. Calculate velocities at mid - step by using

$$v\left(t + \frac{1}{2} dt\right) = v(t) + \frac{1}{2} dt \cdot a(t)$$

3. Calculate $a(t+dt)$
4. Calculate $v(t+dt)$ by

$$v(t+dt) = v\left(t + \frac{1}{2} dt\right) + \frac{1}{2} dt \cdot a(t+dt)$$

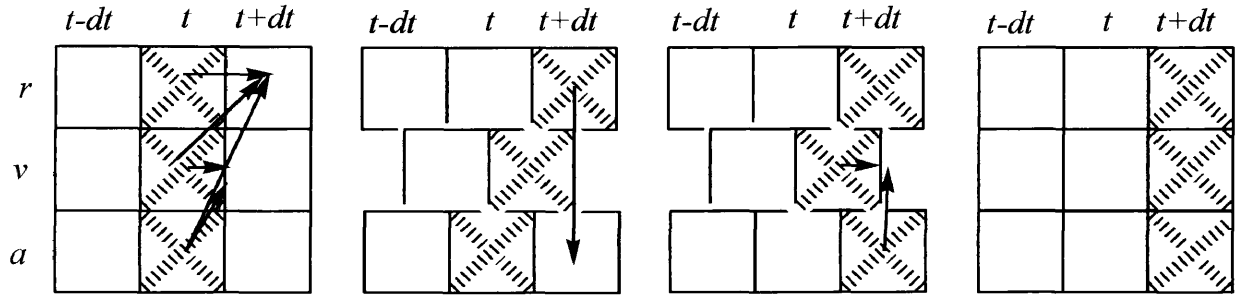


Figure 4. Velocity Verlet Integrator

Figure 4: This figure vividly shows the loop above. Note that this method makes progress via a mid-step calculation. (from M. P. Allen and D. J. Tildesley, 1987. Computer Simulation of Liquids)

This algorithm provides the best evaluation of velocities and requires less computer memory.

2.3.3 Boundary Conditions

Our problem of surface conditions can be addressed by implementing rigid boundary conditions. The two dimensional system may be embedded in a phantom of atoms, i.e., atoms at the boundaries are fixed to their equilibrium positions. When an atom hits the boundary it is bounced back. Rigid boundaries are much easier to code and are more favored in most liquid simulations than are periodic boundary conditions. But in some cases, rigid boundaries introduce artifacts into the system. However the effects will decrease as the system size increase.

2.3.4 Temperature Scaling

To attain the desired temperature, we scale velocities of all atoms:

$$v_{new} = v_{old} \sqrt{\frac{Td}{Ta}}.$$

Here Td is the desired temperature in Kelvins, Ta is the actual temperature computed from the following equation using the old velocities:

$$Ta = \frac{\sum_{i=1}^M m v_i^2}{N_f \cdot k_B},$$

where N_f is the number of degrees of freedom.

From this equation, we can see that temperature is related to the average kinetic energy of the system. i.e.,

$$\langle K \rangle = \frac{N_f k_B T}{2}.$$

In general, the kinetic energy K is: $K = \frac{1}{2} \sum_{i=1}^N m_i v_i^2$.

In our program, N_f is $2N - 1$ because each atom has two velocity components (i.e., v_x and v_y). And one degree of freedom is subtracted because the translation motion is ignored.

This is the simplest form of temperature scaling. We apply this scaling if the actual temperature is higher or lower than the desired temperature by 5K. It is adequate for driving the simulation consistently towards the desired temperature.

2.3.5 Initial Conditions

- Time Step:

To decrease computational time, a large time step should be used. However, too long a time step causes instability and inaccuracy in the integration. The velocity Verlet method usually uses 1 femtosecond (1 fs or 10^{-15} s) or smaller, in order to satisfy fast bond vibrations. We tested the stability of integration process and choose 2×10^{-17} s as our time step. This time step limits our simulations to the order of 1 ns (10^{-9} s) duration.

- Initial Velocities:

A Gaussian distribution of velocities is a good approximation for a given temperature. They help the models to equilibrate in relatively short time. And most importantly, Gaussian random number distributions are very easy to work with.

In an atomic system,

$$\rho(v_{ix}) = \left(\frac{m_i}{2\pi k_B T} \right)^{\frac{1}{2}} e^{-\frac{m_i v_{ix}^2}{2k_B T}}$$

where $\rho(v_{ix})$ is the probability density for velocity component v_{ix} , and similar equation apply for the y component.

The Gaussian-distributed numbers can be obtained from the following steps:

1. Generate 12 uniform random variables, $\xi_1, \xi_2, \dots, \xi_{12}$ in the range $(0, 1)$;

2. Calculate $R = \frac{\sum_{i=1}^{12} \xi_i - 6}{4}$;

3. $\zeta = (((a_9 R^2 + a_7) R^2 + a_5) R^2 + a_3) R^2 + a_1) R$

where $a_1 = 3.949846138$

$$a_3 = 0.252408784$$

$$a_5 = 0.076542912$$

$$a_7 = 0.008355968$$

$$a_9 = 0.029899776$$

This method yields numbers ζ which are sampled from a Gaussian distribution.

2.4 Results and Discussion

The molecular dynamics simulation was tested on several systems of different size and dimension: a $2 \times 2 \times 2$ simple cubic system, a two dimensional 4×4 system, an $8 \times 8 \times 8$ complex cubic system, and an 8×8 two dimensional complex system.

- Figure 5 shows the plot of potential energy vs. time steps of the $2 \times 2 \times 2$ model. This system is driven to the equilibration state after 480,000 time steps. The calculated average positions of all atoms in the whole system are the equilibrium positions.

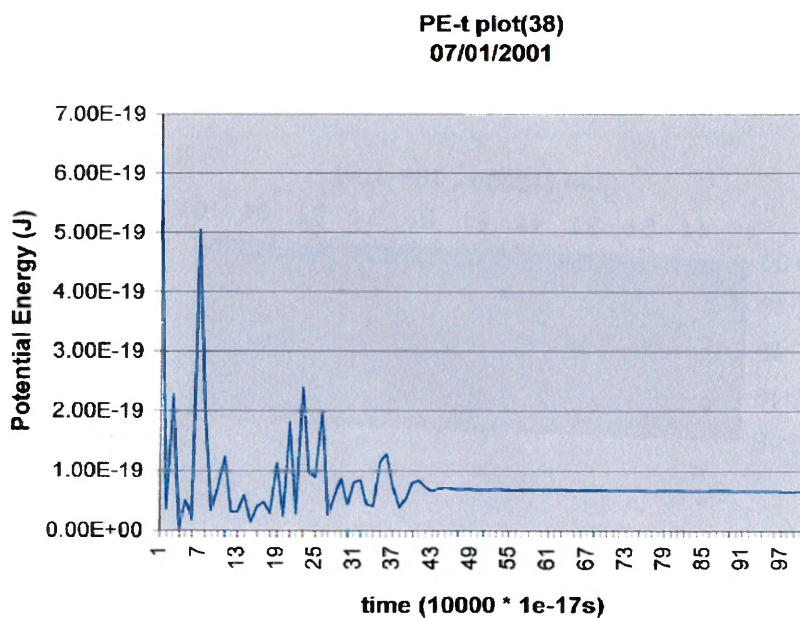


Figure 5.

Figure 5: Fluctuations of three-dimensional small system potential energy during a molecular dynamics simulation,

implying that this model is equilibrated after 480,000 time steps.

- For the two dimensional 4×4 model, we calculated root mean square (RMS) deviations for the amino acid positions for different desired temperatures and force (Hook) constants. We found that the results perfectly agree with the theoretical expectation: the root mean square deviation increases as desired temperature increases and as Hook constant decreases. This is illustrated by Figure 6 - 9.

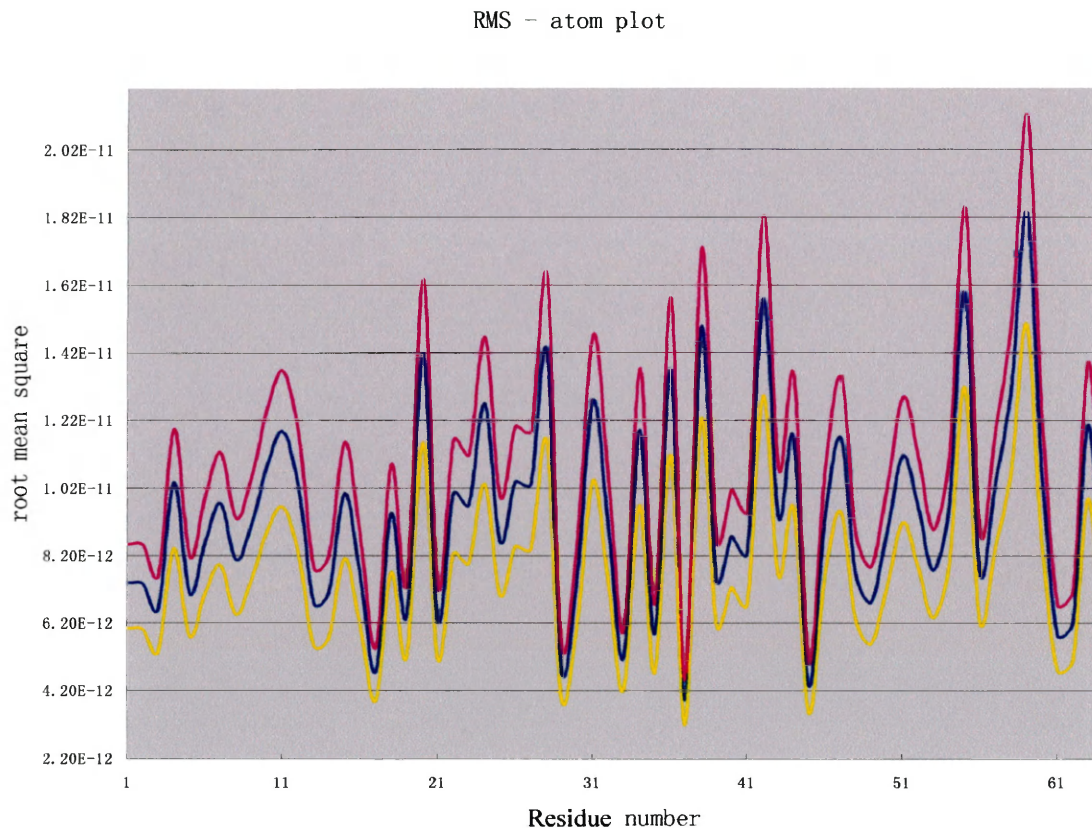


Figure 6

Figure 6: Calculated root mean square deviation of 4×4 model with 3 different desired temperatures. Backbone averages are shown as a function of residue number. a, yellow line, RMS deviation of the system at 198 K; b, blue line, the desired temperature is raised to 298 K; c, pink line, the desired temperature is even higher, 598 K.

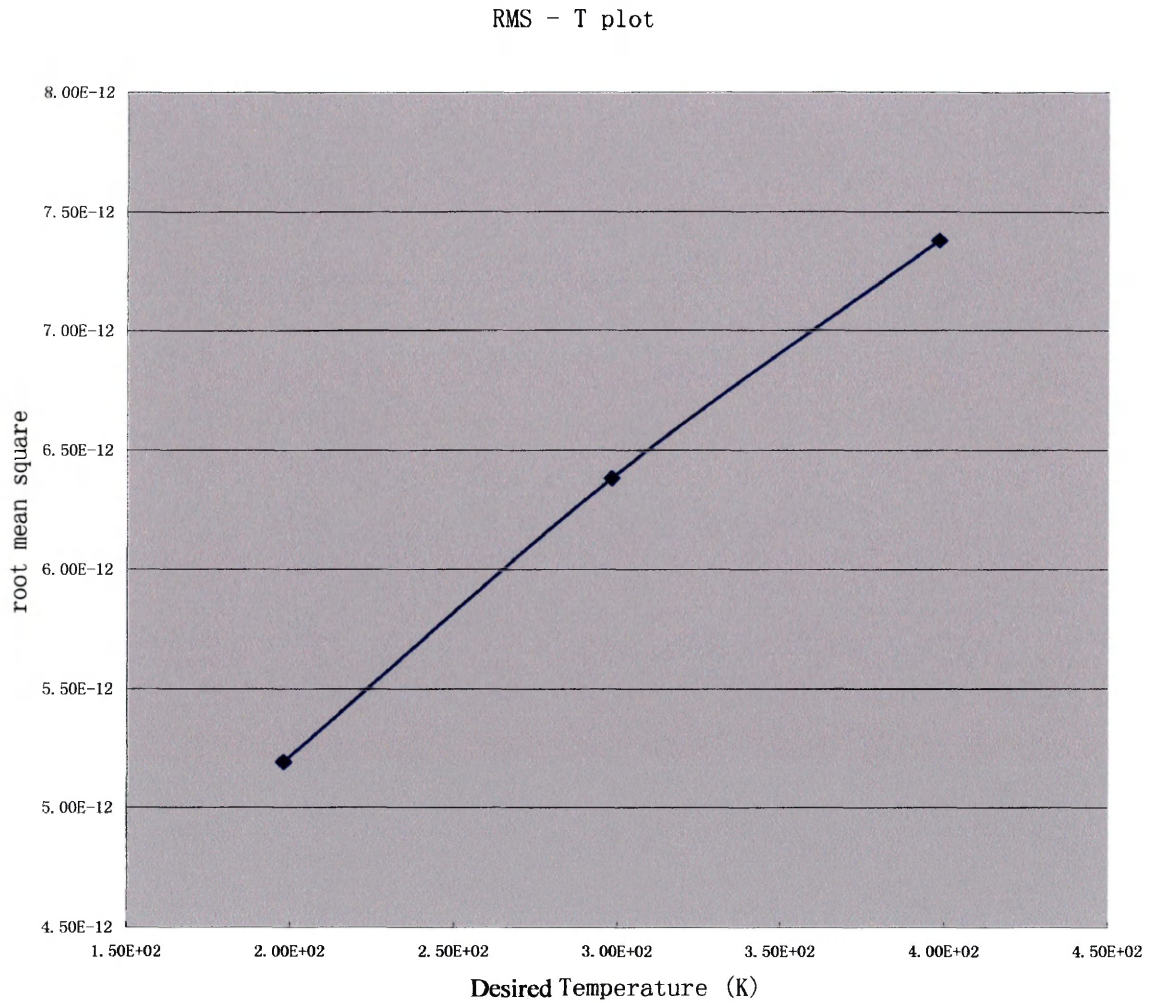


Figure 7

Figure 7: average RMS deviation versus desired temperature, indicating that the higher the desired temperature, the larger the root mean square deviation. This result agrees with a theoretical derivation: since the desired temperature is higher, the Brownian motion is more violent. Consequently, the RMS deviation for each atom's position is larger.

RMS - atom plot

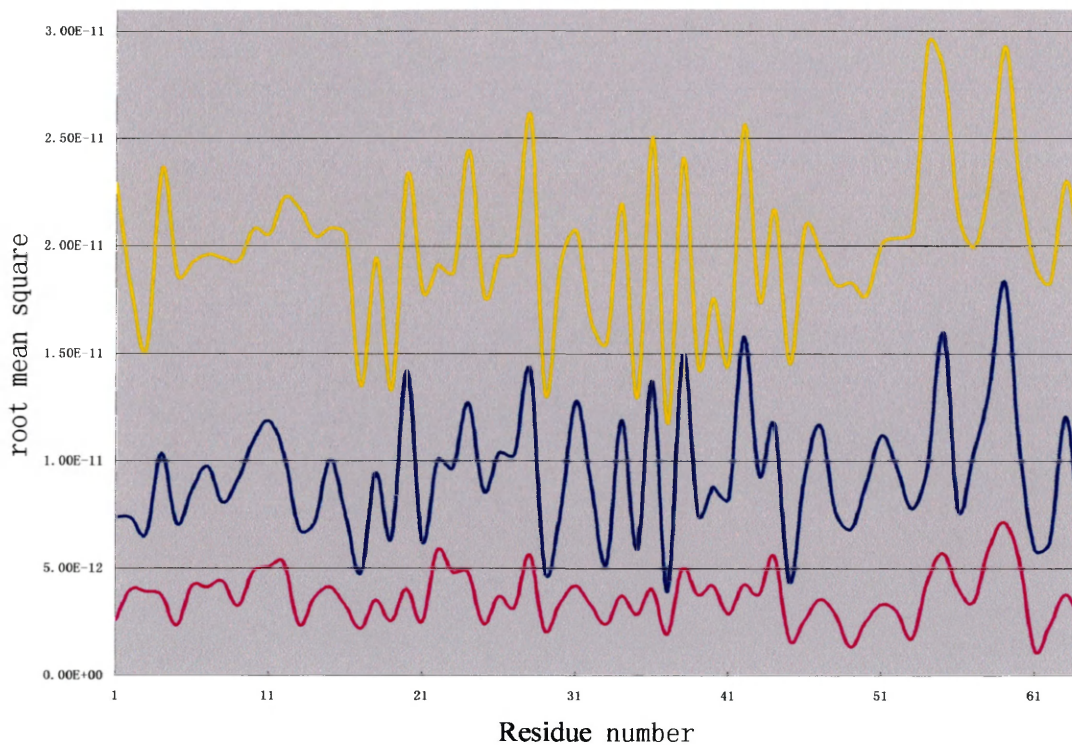


Figure 8

Figure 8: Calculated root mean square deviation of 4×4 model with three different force constants. a, Yellow line, $k_{\text{big}} = k_{\text{small}} = 31.7 \text{ kcal mol}^{-1} \text{ \AA}^{-1}$; b, blue line: $k_{\text{big}} = k_{\text{small}} = 317 \text{ kcal mol}^{-1} \text{ \AA}^{-1}$; c, pink line: $k_{\text{big}} = k_{\text{small}} = 3170 \text{ kcal mol}^{-1} \text{ \AA}^{-1}$. k_{big} and k_{small} are the representations of force constants of stiff and loose interactions, respectively.

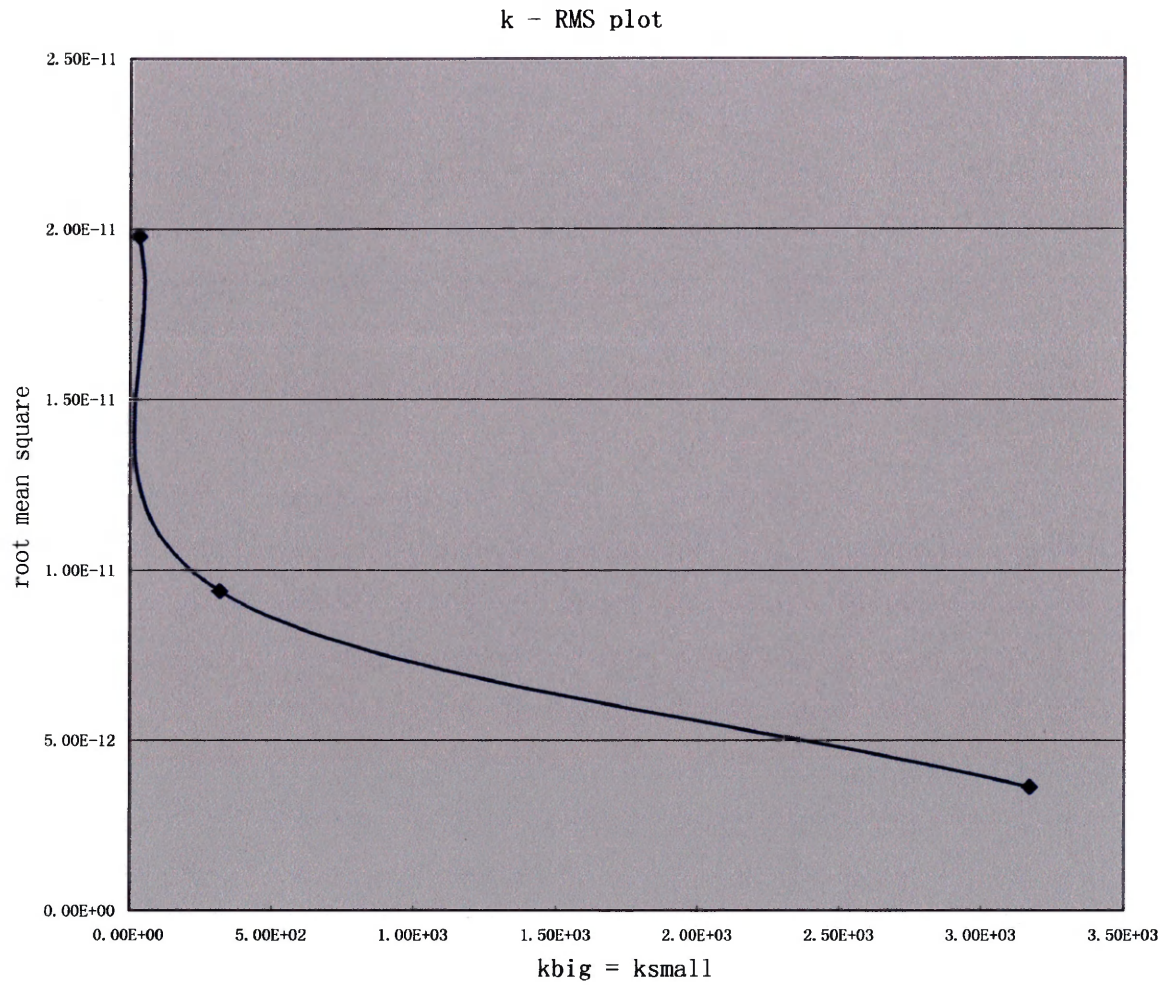


Figure 9

Figure 9: Average root mean square deviation versus force constant. The RMS deviation decreases as force constant increases. Since the force constant is larger, the model is relatively tighter, implying that the RMS deviation is smaller than the initial RMS deviation.

We also studied the relations between potential energy and temperature & force. The results are compared in Table 1 - 3.

Temperature Scaling	Each Step		± 5 K	
Force Constant $k_{\text{big}} = k_{\text{small}}$	220	2.20	220	2.20
Desired Temperature	598 K	598 K	598 K	598 K
Potential Energy (PE)	5.0e-19 to 2.5e-18 J	0 to 3.2e-18 J	1.2e-18 to 1.3e-18 J	1.0e-19 to 2.9e-18 J
Fluctuation of PE	2.0e-18 J small	3.2e-18 J big	0.1e-18 J small*	2.8e-18 J big*

Table 1 – A

Temperature Scaling	Each Step		± 5 K	
Desired Temperature	298 K	598 K	298 K	598 K
Force Constant $k_{\text{big}} = k_{\text{small}}$	220	220	220	220
Potential Energy (PE)	0 to 2e-18 J	5e-19 to 2.5e-18 J	5.9e-19 to 6.4e-19 J	1.2e-18 to 1.3e-18 J
Fluctuation of PE	2e-18 J	2e-18 J	0.5e-19 J	1e-19 J
	Same		almost same compared to * data in Table 1 - A	

Table 1 - B

Desired Temperature	298 K		598 K	
Temperature Scaling	Each Step	± 5 K	Each Step	± 5 K
Force Constant $k_{\text{big}} = k_{\text{small}}$	220	220	220	220
Potential Energy (PE)	0 to 2e-18 J	5.9e-19 to 6.4e-19 J	5.0e-19 to 2.5e-18 J	1.2e-18 to 1.3e-18 J
Fluctuation of PE	20e-19 J big	0.5e-19 J small	20e-19 J big	1e-19 J small

Table 2

Temperature Scaling	Each Step		± 5 K	
Desired Temperature	298 K	598 K	298 K	598 K
Force Constant $k_{\text{big}} = k_{\text{small}}$	220	220	220	220
Potential Energy (PE)	0 to 2e-18 J	5.0e-19 to 2.5e-18 J	5.9e-19 to 6.4e-19 J	1.2e-18 to 1.3e-18 J
Average PE	1e-18 J small	1.5e-18 J big	0.6e-18 J small	1.25e-18 J big

Table 3

In Table 1, the fluctuations of potential energy (PE) are dominated by the value of force constant (k_{big} and k_{small}). While different desired temperatures do not change PE fluctuations much. Table 2 shows that bigger temperature (or kinetic energy) fluctuations make the fluctuations of PE smaller, but do not affect the value of potential energy. Table 3 tested the theory that the value of potential energy is proportional to temperature. The potential energy function is calculated by equation ①.

$$\bar{V} = \frac{\sum_j V_j e^{-\beta V_j}}{\sum e^{\beta V_j}} = \frac{\int \frac{1}{2} kx^2 e^{-\beta \cdot \frac{1}{2} kx^2} dx}{\int e^{-\beta \cdot \frac{1}{2} kx^2} dx} \quad \text{where } \beta = \frac{1}{kT} \quad \text{①}$$

$$\text{let } u = x\sqrt{\frac{1}{2}k}, \text{ then } du = \sqrt{\frac{1}{2}k} dx$$

$$\begin{aligned}
\bar{V} &= \frac{\int u^2 e^{-\beta u^2} du}{\int e^{-\beta u^2} du} \quad \text{let } z = \sqrt{\beta} u \Rightarrow z^2 = \beta u^2 \Rightarrow dz = \sqrt{\beta} du \\
&= \frac{\int \frac{z^2}{\beta} e^{-z^2} \frac{dz}{\sqrt{\beta}}}{\int e^{-z^2} \frac{dz}{\sqrt{\beta}}} = kT \frac{\int z^2 e^{-z^2} dz}{\int e^{-z^2} dz} \propto T
\end{aligned}$$

- Another simulation system is an $8 \times 8 \times 8$ lattice. Again RMS deviations increase as temperature increases and force constant drop. Table 4 lists the average RMS deviation found for two different temperatures and k values (k_{big} and k_{small}).

Desired Temperature	298 K	598 K	298 K
Force Constant $k_{\text{big}} = k_{\text{small}}$	220	220	2.20
Average RMS deviation	0.95e-11 small	1.03e-11 bigger	13.5e-11 huge

Table 4

We calculated RMS deviations in x, y, and z directions and found that the RMS deviations for symmetric positions of symmetric atoms have exactly the same values. e.g.

$$\text{rmsx}[1][1][1] = \text{rmsx}[8][1][1];$$

$$\text{rmsy}[1][1][1] = \text{rmsy}[1][8][1];$$

and `rmsz[1][1][1] = rmsy[1][1][8]`, etc.

- We are now working on the 8×8 model. Figure 9 shows PE vs time step. This system is not equilibrated until 4×10^8 time steps simulations have been completed.

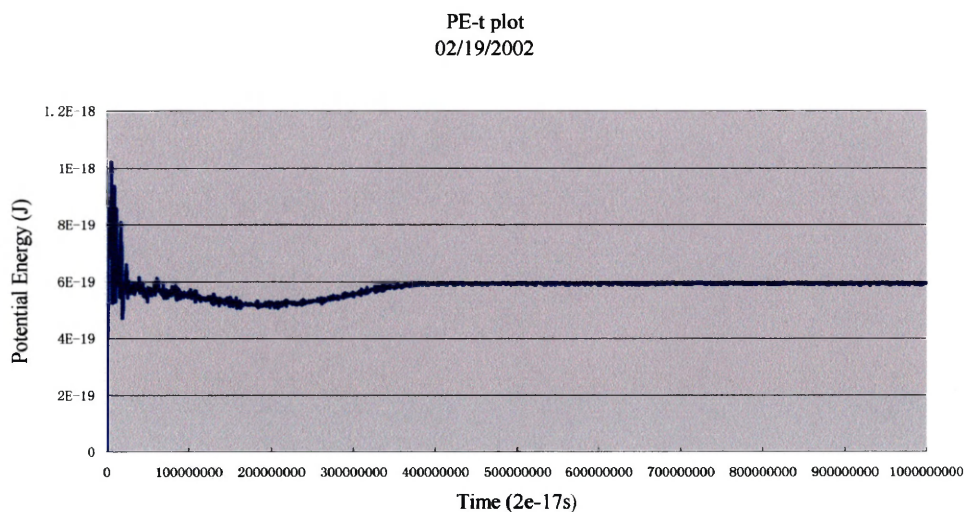


Figure 10

Figure 10: Fluctuation of two-dimensional 8×8 system potential energy during a molecular dynamics simulation, indicating that this system is equilibrated after 400,000,000 time steps.

The following results are concluded from some data collected from unequilibrated systems. They may as yet be

meaningless. But I write them down as a reference for other students in our group.

1. The correlation factor is affected by the values of large and small force constants, and the force constant between the boundary and the enzyme system. The large force constant varies in the range of $1 \leq k_{big} \leq 200$; while the small force constant is in the range of $0.04 \leq k_{small} \leq 40$; and phantom (boundary) force constant in the range of $0.1 \leq k_p \leq 10$. The value of the correlation factor fluctuates, due to the values of force constants, between -0.98 and $+0.98$.

2. A friction factor, which is proportional to the inverse of distance, was added on the catalytic residue to simulate the energy barrier. Consequently, chemical events when the friction factors are high are much less likely than those with low frictional forces. Again, this evidence proves that our program is behaving correctly.

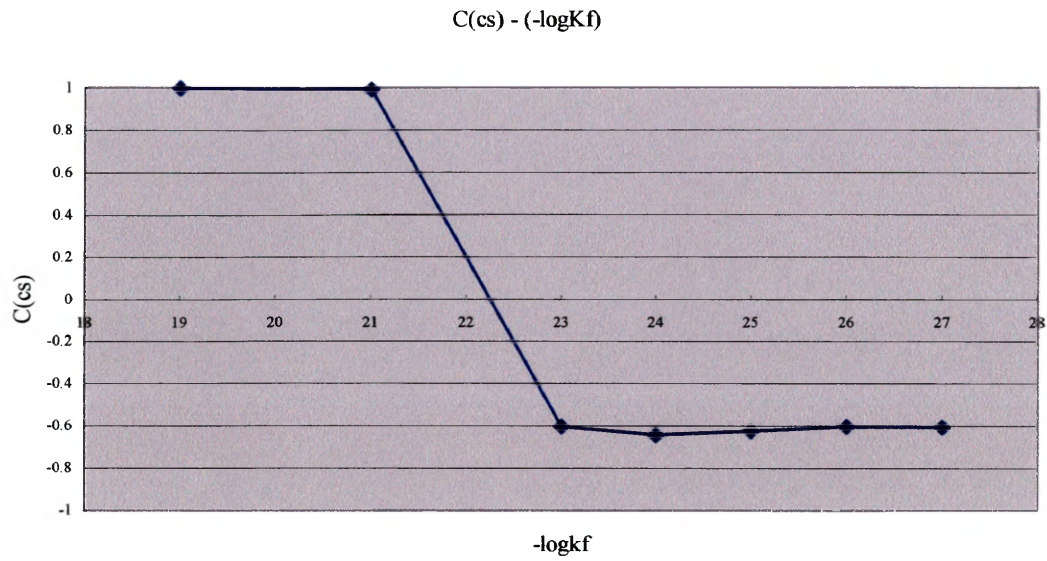


Figure 11

Figure 11: value of correlation factor versus $-\log k_f$, indicating that the chance of chemical events increases as k_f decreases.

3. Most importantly, the correlation factor depends on the distribution of large and small force constants to a large extend. The range varies from 0.121 to 0.871 for only 5 random-generated distributions.

Next, we should address the question of the optimal Hook constant, temperature, and friction factor (k_f). Other students

in our group will achieve further progress in investigating this program and intersect it with genetic algorithms.

Conclusion

An MD simulation method to calculate chemical efficiency of a simplified enzyme model has been tested and is ready to be optimized. This method is based on using the velocity Verlet algorithm, as an integrator for Newton's equations of motion. The reliability of the method is mainly attributed to our calculations of RMS deviations, potential energy fluctuations, and correlation factors. This is demonstrated by the success of 4 different size and dimension models. The accuracy of this MD program, for the 8×8 model, is now ready for optimization.

Work to extend the method and to intersect it with the genetic algorithms in order to treat our cases is underway. Most importantly, we need to find an efficient way to use the simulations, due to CPU time limitations.

Part II

Genetic Algorithms

Chapter 3

Genetic Algorithms

Genetic algorithms (GAs) were invented by John Holland in 1975 (Holland 1975). They work very well in finding the global optimal solution in complex search problems. Genetic algorithms are based on the Natural (Darwinian) Selection. The basic idea of genetic algorithms is to give preference to fitter species and allowing them to pass on their genes to the next generation. Many real world problems involve search and optimization and are ideal for genetic algorithms. But these algorithms are computationally expensive compared to other methods.

At the start of our optimization, an initial group parents each with a different number and distribution of large and small force constants is generated randomly (Initialization). Then we compute and save the correlation factor for each individual in the current group. Each member of this initial group is evaluated by their correlation factors (Evaluation). To form the next generation, we reproduce more

copies of individuals whose correlation factors are more negative (Selection). Lastly, we introduce new individuals by “Mutation” and “Crossover”: in the “Mutation” procedure, we swap large force constants with small force constants at random with some low probability. This step helps us prevent premature convergence and induce a random walk through the search space. “Crossovers” are used to create two new “offspring” by recombining parts of promising “parents” as follow.

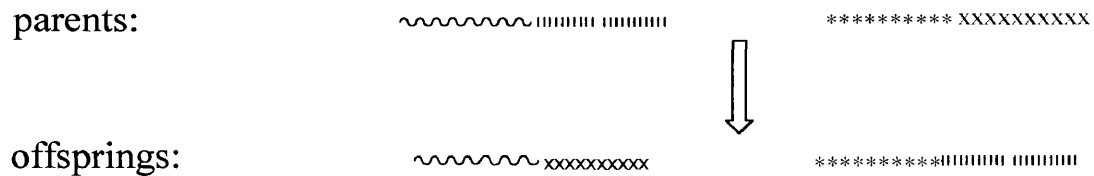


Figure 12: Crossover Operator

This operator produces two individuals that replace the old parents. During this procedure, a random walk is generated through the whole system. The flowchart and pseudo code are as follow:

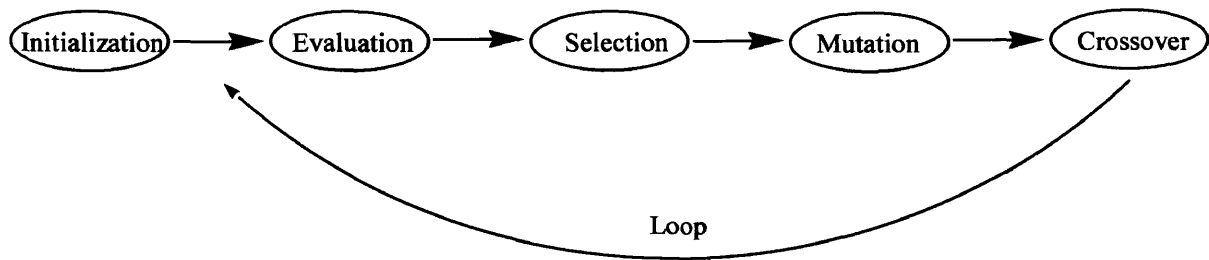


Figure 13: Genetic Algorithms

Pseudo Code:

Begin GA

g = 0; // generation counter

Initialization P(g);

Evaluation P(g); // i.e., compute fitness values

while (!done)

{

 g++;

 Selection (P(g) from P(g-1));

 Mutation P(g);

 Crossover P(g);

 Evaluation P(g);

};

end of GA

Part III

Appendices

Appendix A

How to Use?

Compile command:

```
g++ -g -Wall main.cc initialize.cc scale.cc -o main
```

Then, run main and wait

We can check out the initial conditions such as initial positions from “InitR.out”; initial velocities from “InitV.out”; and force constant from “InitKx.out” and “InitKy.out”. After your finishing the simulation, you can get the information such as potential energies for each “print” steps from “PE.out”; final positions from “R.out”; final velocities, average positions, correlation factors from “V.out”; and positions of C , S , F for each “print” steps from “RxC.out”, “RyC.out”, “Rxs.out”, “Rys.out”, “RxF.out”, and “RyF.out”.

Appendix B

Verlet.h

```
//*****  
// verlet.h  
// Purpose: header file  
// Arthor: Shiyang Shang  
// Advisor: Carey K. Bagdassarian  
//*****  
  
const int n1 = 10;           // 10 atoms on x direction  
const int n2 = 10;           // 10 atoms on y direction  
const int nAtom = n1*n2;     // 10 by 10 atoms totally  
const long double m = 12.01/(6.02214e26);  
    // the weight of particle i (Kg/atom)  
const long double c = 1.523e-10;  
    // the expected distance between atoms (m/atom)  
const long double dt = 2e-17;  
    // time step: dt = 10-2 (fs) = 10-17  
const double A_Large_Number = 4e8;  
    // the time steps using in the Verlet algorithm  
const int N = (n1-2)*(n2-2); // 8 by 8 atoms actually  
const double Td = 298; // desired temperature
```

```

const long double KB = 1.38066e-23;      //(J/K)
const long double ksmall = 0.4;
const long double kbig = 20.0;
const double print = 4e5;      // collect data every 4e5 steps
const double rmsstep = 1;
const double cutoff = 0; // 6000000;
const int Cx = 4;
const int Cy = 4;
//const int Cz = 1;
const int Sx = 5;
const int Sy = 4;
//const int Sz = 1;
const int Fx = 4;
const int Fy = 5;
//const int Fz = 1;
//const int Nx = 2;
//const int Ny = 2;

const long double kf = 1.4e-23;
typedef long double Position[n1][n2];
typedef long double Velocity[n1][n2];
typedef long double K[n1][n2];

#include <cmath>
#include <fstream.h>          // for file I/O
#include <iomanip.h>          // for reporting data

```

```

void InitializeData(Position, Position, Position, Position,
                    Position, Position, Position, Position,
                    Velocity, Velocity, Velocity, Velocity,
                    K, K);

void RandPosition(Position, Position, Position, Position,
                  Position, Position);

void RandVelocity(Velocity, Velocity, long double&);

void ReadK(K, K);

void SumSq(Position, Position, Position, Position, Position,
           Position, Position);

void Verlet(Position, Position, Position, Position, Position,
            Position, Position, Position, Velocity, Velocity,
            Velocity, Velocity, K, K, long double&, long double,
            Position, Position, double&, double&, double&, long
            double&, long double&);

void Temp(Velocity, Velocity, double);

void Report(ofstream&, ofstream&, ofstream&, ofstream&,
            ofstream&, ofstream&, ofstream&, Position, Position,

```



```
long double);
```

Appendix C

main.cc

```
//*****
// main.cc
//      Purpose:  main function
//      Author:   Shiyang Shang
//      Advisor:  Carey Bagdassarian
//*****

#include "verlet.h"

int main(void)
{
    Position Rx;          // positions on x direction
    Position Ry;          // positions on y direction
    //      Position Rz; // positions on z direction (3D only)
    Position Rx0;         // initial positions on x
    Position Ry0;         // initial positions on y
    //      Position Rz0; // initial positions on z (3D)
    Position AvgRx;       // average positions on x
    Position AvgRy;       // average positions on y
    //      Position AvgRz; // average positions on z (3D)
    Position RxNew;       // new positions on x
    Position RyNew;       // new positions on y
    //      Position RzNew; // new positions on z (3D)
    Position rmsx;         // root mean squares on x
```

```

Position rmsy;          // root mean squares on y
//    Position rmsz; // root mean squares on z (3D)
Velocity Vx;           // velocities on x
Velocity Vy;           // velocities on y
//    Velocity Vz;  // velocities on z (3D)
Velocity VxMid;        // mid-step velocities on x
Velocity VyMid;        // mid-step velocities on y
//    Velocity VzMid; // mid-step velocities on z (3D)
K Kx;                  // hook constant on x
K Ky;                  // hook constant on y
//    K Kz;           // hook constant on z (3D)
long double SumVsqr = 0.0; // summation of  $v^2$ 
long double PE = 0.0;     // potential energy
long double KE = 0.0;     // kinetic energy
//    long double TOT = 0.0; // total energy PE + KE
//    long double counter = 0.0;
// counter of chemical events
//    long double dcs = 2.0 * c;
// desired distance between catalytic amino acid ( $C$ ) and
// substrate ( $S$ )
//    long double dcsNew = 0.0;
// actual distance between  $C$  and  $S$ 

double Cij1 = 0.0;        //  $\sum_{i=1}^M [(X_s - \overline{X_s})(X_c - \overline{X_c})]_i$ 

double Cij2 = 0.0;        //  $\sqrt{\sum_{i=1}^M (X_c - \overline{X_c})_i^2}$ 

```

```

double Cij3 = 0.0;           //  $\sqrt{\sum_{i=1}^M (X_s - \overline{X_s})_i^2}$ 

double Ta;                   // actual temperature

ofstream outFile;            // output file

ofstream outFile1;

ofstream outFile2;

ofstream outFile3;

ofstream outFile4;

//    ofstream outFile5;

ofstream outFile6;

ofstream outFile7;

//    ofstream outFile8;

ofstream outFile9;

ofstream outFile10;

//    ofstream outFile11;


outFile.open ( "V.out" );// output velocities of final step
outFile.precision(3);      // precision is 3
outFile1.open ("R.out"); // output positions of final step
outFile1.precision(2);

outFile2.open ( "PE.out" );

    // output potential energies every "print" steps
outFile2.precision(3);

outFile3.open ("RxC.out");

    // output positions of C on x every "print" steps
outFile3.precision(3);

```

```

outFile4.open ("RyC.out");
    // output positions of  $C$  on  $y$  every "print" steps
outFile4.precision(3);
outFile6.open ("RxF.out");
    // output positions of  $F$  on  $x$  every "print" steps
outFile6.precision(3);
outFile7.open ("RyF.out");
    // output positions of  $F$  on  $y$  every "print" steps
outFile7.precision(3);
outFile9.open ("Rxs.out");
    // output positions of  $S$  on  $x$  every "print" steps
outFile9.precision(3);
outFile10.open ("RyS.out");
    // output positions of  $S$  on  $y$  every "print" steps
outFile10.precision(4);

InitializeData(Rx, Ry, Rx0, Ry0, AvgRx, AvgRy, rmsx, rmsy,
              Vx, Vy, VxMid, VyMid, Kx, Ky);

RandPosition(Rx, Ry, Rx0, Ry0, RxNew, RyNew);

RandVelocity(Vx, Vy, SumVsqr);

ReadK(Kx, Ky);

for(int j = n2-2; j > 0; j--)

```

```

for(int i = 1; i < n1-1; i++)
{
    PE = PE + 0.5 * Kx[i][j] * pow((Rx[i+1][j] - Rx[i][j] -
        c), 2) + 0.5 * Kx[i-1][j] * pow((Rx[i][j] -
        Rx[i-1][j] - c), 2) + 0.5 * Ky[i][j] *
        pow((Ry[i][j+1] - Ry[i][j] - c), 2) + 0.5 * Ky[i][j-
        1] * pow((Ry[i][j] - Ry[i][j-1] - c), 2);

// potential energy:  $V = \sum_{i=1}^n \left[ \frac{1}{2}(x_i - x_{i-1})^2 + \frac{1}{2}(x_i - x_{i+1})^2 + \frac{1}{2}(y_i - y_{i-1})^2 + \frac{1}{2}(y_i - y_{i+1})^2 \right]$ 
//      KE = KE + 0.5 * m * Vx[i][j][k] * Vx[i][j][k];
}
//      TOT = PE + KE;

Report(outFile2, outFile3, outFile4, outFile6, outFile7,
        outFile9, outFile10, Rx, Ry, PE);

double i = 1;
while (i < A_Large_Number+1)
{
    Verlet(Rx, Ry, Rx0, Ry0, RxNew, RyNew, AvgRx, AvgRy, Vx,
        Vy, VxMid, VyMid, Kx, Ky, SumVsqr, i, rmsx, rmsy,
        Cij1, Cij2, Cij3, PE, KE);
//      if ( (Vx[Cx][Cy][Cz] > 0) && (Vx[Sx][Sy][Sz] < 0) )
//          {
//              dcsNew = fabs(Rx[Sx][Sy][Sz]-Rx[Cx][Cy][Cz]);
//              if ( dcsNew < dcs )

```

```

//          dcs = dcsNew;
//          else
//          {
//              counter = counter + 1;
//              outFile << "          " << Vx[Cx][Cy][Cz]
//                  << "          " << Vx[Sx][Sy][Sz]
//                  << "          " << dcs/c << endl;
//              dcs = 2*c;
//          }
//      }

```

$$Ta = \text{SumVs}q * m / ((2 * N - 1) * KB); \quad // \quad Ta = \frac{\sum_{i=1}^M m v_i^2}{N_f \cdot K_B}$$

```

if (i/print == int(i/print) && i >= cutoff)
    Report(outFile2, outFile3, outFile4, outFile6, outFile7,
           outFile9, outFile10, Rx, Ry, PE);
if ((Ta-Td) > 5.0 || (Ta-Td) < -5.0) //temperature scaling
{
    Temp(Vx, Vy, Ta);
}
if (i == A_Large_Number) // print out final step condition
{
    outFile << "          V (final): " << endl;
    outFile1 << "          R (final): " << endl;
    for (int b = n2-1; b > -1; b--)
    {

```

```

        for (int a = 0; a < n1; a++)
        {
            outFile << "      (" << Vx[a][b] << ", "
                << Vy[a][b] << ")";
            outFile1 << "      (" << Rx[a][b] << ", "
                << Ry[a][b] << ")";

        }
        outFile << endl;
        outFile1 << endl;
    }

    i = i + 1;
}

outFile << endl << "          Average Position: " << endl;
// print out average positions for all atoms
for (int j = n2-1; j > -1; j--)
{
    for (int i = 0; i < n1; i++)
        //          for (int k = 1; k < n-1; k++)
        {
            AvgRx[i][j] = AvgRx[i][j]/(A_Large_Number-cutoff + 1);
            AvgRy[i][j] = AvgRy[i][j]/(A_Large_Number-cutoff + 1);
            //          AvgRz[i][j] = AvgRz[i][j]/(A_Large_Number -
            //          cutoff + 1);
            outFile << "      (" << AvgRx[i][j] << ", "

```



```

        << AvgRy[i][j] << ")";
    }
    outFile << endl;
}
outFile << endl;
outFile << "                " << "Ci j = "
        << (Cij1/(sqrt(Cij2)*sqrt(Cij3))) << endl;

//    outFile << "                Root Mean Square: " << endl;
//    for (int i = 1; i < n-1; i++)
//        for (int j = 1; j < n-1; j++)
//            for (int k = 1; k < n-1; k++)
//                outFile << "                "
//<<< sqrt(rmsx[i][j][k]/((A_Large_Number-cutoff+1)/rmsstep))
//                << "                "
//<<< sqrt(rmsy[i][j][k]/((A_Large_Number-cutoff+1)/rmsstep))
//                << "                "
//<<< sqrt(rmsz[i][j][k]/((A_Large_Number-cutoff+1)/rmsstep))
// << endl;

//    outFile << "                " << "Counter = " << counter;

outFile.close();
outFile1.close();
outFile2.close();
outFile3.close();

```

```
    outFile4.close();  
    outFile6.close();  
    outFile7.close();  
    outFile9.close();  
    outFile10.close();  
  
    return 0;  
} // end of main()
```

Appendix D

Initialize.cc

```
//*****  
// Initialize.cc  
//      Purpose: includes all functions related to the initial  
//                  conditions  
//      Author:  Shiyong Shang  
//      Advisor: Carey K. Bagdassarian  
//*****  
#include "verlet.h"  
  
//*****  
// void InitializeData(Position Rx, Position Ry, Position Rz,  
//                      Velocity Vx, Velocity Vy, Velocity Vz,  
//                      K Kx, K Ky, K Kz)  
//      Purpose: Initialize all data to 0.  
//*****  
void InitializeData(Position Rx, Position Ry,  
                   Position Rx0, Position Ry0,  
                   Position AvgRx, Position AvgRy,  
                   Position rmsx, Position rmsy,  
                   Velocity Vx, Velocity Vy,  
                   Velocity VxMid, Velocity VyMid,
```

```

        K Kx, K Ky)
{
    for (int j = n2-1; j > -1; j--)
        for(int i = 0; i < n1; i++)
        {
            Rx[i][j] = 0.0;      //Intialize all data to 0
            Ry[i][j] = 0.0;
            //          Rz[i][j][k] = 0.0;

            Rx0[i][j] = 0.0;
            Ry0[i][j] = 0.0;
            //          Rz0[i][j][k] = 0.0;

            AvgRx[i][j] = 0.0;
            AvgRy[i][j] = 0.0;
            //          AvgRz[i][j][k] = 0.0;

            Vx[i][j] = 0.0;
            Vy[i][j] = 0.0;
            //          Vz[i][j][k] = 0.0;

            Kx[i][j] = 10.0;
            Ky[i][j] = 10.0;
            //          Kz[i][j][k] = ksmall;

            VxMid[i][j] = 0.0;

```

```

        VyMid[i][j] = 0.0;
        //          VzMid[i][j][k] = 0.0;

        rmsx[i][j] = 0.0;
        rmsy[i][j] = 0.0;
        //          rmsz[i][j][k] = 0.0;
    }

    return;
} // end of InitializeData


//*****
// void RandPosition(Position Rx, Position Ry, Position Rz)
//    Purpose:  initializes positions of atoms to the vicinity
//              of equilibrium position
//*****
void RandPosition(Position Rx, Position Ry,
                  Position Rx0, Position Ry0,
                  Position RxNew, Position RyNew)
{
    //    long double Rand[n][n];
    ifstream inFile;
    ofstream outFile;
    //    inFile.open("R.dat"); // read data from "R.dat"
    outFile.open("InitR.out"); // write data to "InitR.out"

```

```

outFile.precision(4);

for(int j = n2-1; j > -1; j--)
    for(int i = 0; i < n1; i++)
        {
            Rx[i][j] = (i + 0.5)*c;
            Ry[i][j] = (j + 0.5)*c;
            //          Rz[i][j][k] = (k + 0.5)*c;
        }

//    Rx[Cx][Cy] = (Cx+0.25)*c;
//    Rx[Sx][Sy] = (Sx+0.75)*c;
// activate if you want initial positions are 25% off center

//    for(int i = 1; i < n-1; i++)
//        for(int j = 1; j < n-1; j++)
//            for(int k = 1; k < 2; k++)
//                inFile >> Rx[i][j] >> Ry[i][j];
// activate if you want to read initial positions from a file

for (int j = n2-1; j > -1; j--)
    {
        for (int i = 0; i < n1; i++)
            {
                outFile << "    (" << Rx[i][j] << ", "
                    << Ry[i][j] << ")";
            }
    }

```

```

        Rx0[i][j] = (i + 0.5)*c;
        Ry0[i][j] = (j + 0.5)*c;
// initial positions are all at the center of their boxes
        RxNew[i][j] = Rx0[i][j];
        RyNew[i][j] = Ry0[i][j];
    } // for j
    outFile << endl;
} // for i

//    inFile.close();
outFile.close();
return;
} // end of RandPosition

//*****
// void RandVelocity(Velocity Vx, Velocity Vy, Velocity Vz)
//    Purpose: Initalize Guassian distributed velocities
//*****
void RandVelocity(Velocity Vx, Velocity Vy,
                 long double& SumVsqr)
{
    long double RealSumVsqr;
    long double q;                // temperature scale factor

```

```

ifstream inFile;
ofstream outFile;

inFile.open("Guassian.dat"); // read from "Guassian.dat"
outFile.open("InitV.out"); // write to "InitV.out"

for(int j = n2-2; j > 0; j--)
    for(int i = 1; i < n1-1; i++)
    {
        inFile >> Vx[i][j] >> Vy[i][j];
        SumVsqr = SumVsqr + pow(Vx[i][j], 2)
            + pow(Vy[i][j], 2);
    } // for

RealSumVsqr = Td*(2*N-1)*KB/m;
q = sqrt(RealSumVsqr/SumVsqr);

outFile.precision(3);

for(int j = n2-1; j > -1; j--)
{
    for(int i = 0; i < n1; i++)
    {
        Vx[i][j] = q*Vx[i][j];
        Vy[i][j] = q*Vy[i][j]; // velocities after scaling
        // Vz[i][j][k] = q*Vz[i][j][k];
        outFile << "          (" << Vx[i][j] << ", "

```



```

        << Vy[i][j] << " ";
    } // for j
    outFile << endl;
} // for i

SumVsqr = RealSumVsqr; // summation of  $mv^2$ 

inFile.close();
outFile.close();
return;
} // end of RandVelocity

//*****
// void ReadK(K Kx, K Ky, K Kz)
//Purpose: Initialize the hook constant according to the aa.dat
//*****
void ReadK(K Kx, K Ky)
{
    int aa[n1][n2];
    ifstream inFile;
    ofstream outFile;
    ofstream outFile1;

```

```

inFile.open("aa.dat");      // read aa seq. from "aa.dat"
outFile.open("InitKx.out");// write Kx to "InitKx.out"
outFile1.open("InitKy.out"); // write Ky to "InitKy.out"

for ( int j = n2-1; j > -1; j--)
    for ( int i = 0; i < n1; i++ )
        aa[i][j] = 0;    // initialize all aa to 0

for ( int j = n2-2; j > 0; j-- )
    for ( int i = 1; i < n1-1; i++ )
        inFile >> aa[i][j]; // all aa are in form of 1 or 0

for ( int j = n2-2; j > 0; j-- )
    for ( int i = 1; i < n1-2; i++ )
    {
        if ( aa[i][j] != aa[i+1][j] )
            Kx[i][j] = kbig;
        // hook constant is kbig if it' s between different type of aa
        else
            Kx[i][j] = ksmall;
        // hook constant is ksmall if it' s between same type of aa
    }

for ( int i = 1; i < n1-1; i++ )
    for ( int j = 1; j < n2-2; j++ )
    {

```

```

        if ( aa[i][j] != aa[i][j+1] )
            Ky[i][j] = kbig;
        else
            Ky[i][j] = ksmall;
    }

    outFile.setf(ios::fixed, ios::floatfield);
    // Set up double pt.
    outFile.precision(1);
    outFile1.setf(ios::fixed, ios::floatfield);
    outFile1.precision(1);
    for(int j = n2-1; j > -1; j--)
    {
        for(int i = 0; i < n1-1; i++)
            outFile << "      " << Kx[i][j];
        outFile << endl;
    }

    for(int j = n2-2; j > -1; j--)
    {
        for(int i = 0; i < n1; i++)
            outFile1 << "      " << Ky[i][j];
        outFile1 << endl;
    }

    inFile.close();

```

```
        outFile.close();  
        return;  
    } // end of ReadK
```

Appendix E

scale.cc

```
//*****
// scale.cc
//      Purpose: includes all function related to the velocity
//                  scaling
//      Author:  Shiyong Shang
//      Advisor: Carey K. Bagdassarian
//*****

#include "verlet.h"

//*****
// void verlet(Position Rx, Position Ry, Position Rz,
//              Velocity Vx, Velocity Vy, Velocity Vz,
//              K Kx, K Ky, K Kz)
//      Purpose:  This program allows the user to compute the //
//                  new positions and velocities of particles by
//                  using the Velocity Verlet Algorithm.
```

```

//*****
void Verlet(Position Rx, Position Ry,
            Position Rx0, Position Ry0,
            Position RxNew, Position RyNew,
            Position AvgRx, Position AvgRy,
            Velocity Vx, Velocity Vy,
            Velocity VxMid, Velocity VyMid,
            K Kx, K Ky, long double& SumVsqr, long double step,
            Position rmsx, Position rmsy, double& Cij1,
            double& Cij2, double& Cij3, long double& PE,
            long double& KE)
{
    long double Ax = 0.0, Ay = 0.0;
        // accelerations at time t:  $a(t)$ 
    long double AxNew = 0.0, AyNew = 0.0;
        // accelerations at time t+dt:  $a(t+dt)$ 
    long double VxNew = 0.0, VyNew = 0.0;
        // velocities at time t+dt:  $v(t+dt)$ 
    long double SumVxsqr = 0.0, SumVysqr = 0.0;
        // summation of  $mv_x^2$  and  $mv_y^2$ 
    long double EqR;    // equilibration position of C
    long double EqRS;   // equilibration position of S
    long double dcf = 0.0; //distance between F and C
    SumVsqr = 0.0;      // initialize sum of  $v^2$  to 0
    PE = 0.0;           // initialize potential energy to 0

```

```

for (int j = n2-2; j > 0; j--)
{
    for (int i = 1; i < n1-1; i++)
    {
        dcf = fabs(Ry[Fx][Fy]-Ry[Cx][Cy]);
        if (i == Cx && j == Cy && dcf < c && Vx[Cx][Cy] > 0
            && Vx[Sx][Sy] < 0)
        {
            Ax = (Kx[i][j] * (Rx[i+1][j] - Rx[i][j] - c)
                - Kx[i-1][j] * (Rx[i][j] - Rx[i-1][j] - c) -
                kf/dcf)/m;
        }
        else if (i == Fx && j == Fy && dcf < c && Vx[Cx][Cy]
            > 0 && Vx[Sx][Sy] < 0)
        {
            Ax = (Kx[i][j] * (Rx[i+1][j] - Rx[i][j] - c)
                - Kx[i-1][j] * (Rx[i][j] - Rx[i-1][j] - c)
                + kf/dcf)/m;
        }
        else
            Ax = (Kx[i][j] * (Rx[i+1][j] - Rx[i][j] - c)
                - Kx[i-1][j] * (Rx[i][j] - Rx[i-1][j] - c))/m;
        RxNew[i][j] = Rx[i][j] + dt * Vx[i][j]
            + 0.5 * dt * dt * Ax;
        Rx[i][j] = RxNew[i][j];
    }
}

```

```

    if (step >= cutoff)
        AvgRx[i][j] = AvgRx[i][j] + Rx[i][j];
        VxMid[i][j] = Vx[i][j] + 0.5 * dt * Ax;
    }
for (int i = 1; i < nl-1; i++)
{
    if (step >= cutoff &&
        (step/rmsstep) == int(step/rmsstep))
    {
        //          EqR = (0.5 + i) * c;
        //          rmsx[i][j][k] = rmsx[i][j][k] +
        //          pow((Rx[i][j][k] - EqR), 2);
        if ( i == nl-2 && j == 1 )
        {
            EqR = (0.5 + Cx) * c;
            EqRS = (0.5 + Sx) * c;
            Cij1 = Cij1 +
                ( (Rx[Cx][Cy]-EqR)*(Rx[Sx][Sy]-EqRS) );
            Cij2 = Cij2 + pow((Rx[Cx][Cy] - EqR), 2);
            Cij3 = Cij3 + pow((Rx[Sx][Sy] - EqRS), 2);
        }
    }
    if (step >= cutoff &&
        (step/print) == int(step/print))
        PE = PE + 0.5 * Kx[i][j] * pow((Rx[i+1][j]

```



```

        - Rx[i][j] - c), 2) + 0.5 * Kx[i-1][j] *
        pow((Rx[i][j] - Rx[i-1][j] - c), 2);
dcf = fabs(Ry[Fx][Fy]-Ry[Cx][Cy]);
if (i == Cx && j == Cy &&
    dcf < c && VxMid[Cx][Cy] > 0 && VxMid[Sx][Sy] < 0)
{
    AxNew = (Kx[i][j] * (RxNew[i+1][j]
        - RxNew[i][j] - c) - Kx[i-1][j] * (RxNew[i][j]
        - RxNew[i-1][j] - c) - kf/dcf)/m;
}
else if (i == Fx && j == Fy && dcf < c &&
    VxMid[Cx][Cy] > 0 && VxMid[Sx][Sy] < 0)
{
    AxNew = (Kx[i][j] * (RxNew[i+1][j] -
        RxNew[i][j] - c) - Kx[i-1][j] * (RxNew[i][j] -
        RxNew[i-1][j] - c) + kf/dcf)/m;
}
else
AxNew = (Kx[i][j] * (RxNew[i+1][j] - RxNew[i][j]
    - c) - Kx[i-1][j] *
    (RxNew[i][j] - RxNew[i-1][j] - c))/m;
VxNew = VxMid[i][j] + 0.5 * dt * AxNew;
Vx[i][j] = VxNew;
SumVxsq = SumVxsq + pow(VxNew, 2);
}
}

```

```

for( int i = 1; i < n1-1; i++)
    //      for( int k = 1; k < 2; k++) // (3D)
    {
        for (int j = 1; j < n2-1; j++)
            {
                Ay = (Ky[i][j] * (Ry[i][j+1] - Ry[i][j] - c)
                    - Ky[i][j-1] * (Ry[i][j] - Ry[i][j-1] - c))/m;
                RyNew[i][j] = Ry[i][j] + dt * Vy[i][j] +
                    0.5 * dt * dt * Ay;
                Ry[i][j] = RyNew[i][j];
                if (step >= cutoff)
                    AvgRy[i][j] = AvgRy[i][j] + Ry[i][j];
                VyMid[i][j] = Vy[i][j] + 0.5 * dt * Ay;
            }
        for (int j = 1; j < n2-1; j++)
            {
                //      if (step >= cutoff &&
                //      (step/rmsstep) == int(step/rmsstep))
                //      {
                //      EqR = (0.5 + j) * c;
                //      rmsy[i][j][k] = rmsy[i][j][k]
                //      + pow((Ry[i][j][k] - EqR), 2);
                //      }
                if (step >= cutoff &&
                    (step/print) == int(step/print))

```

```

        PE = PE + 0.5 * Ky[i][j] * pow((Ry[i][j+1]
            - Ry[i][j] - c), 2) + 0.5 * Ky[i][j-1] *
            pow((Ry[i][j] - Ry[i][j-1] - c), 2);
        AyNew = (Ky[i][j] * (RyNew[i][j+1] -
            RyNew[i][j] - c) - Ky[i][j-1] * (RyNew[i][j] -
            RyNew[i][j-1] - c))/m;
        VyNew = VyMid[i][j] + 0.5 * dt * AyNew;
        Vy[i][j] = VyNew;
        SumVysq = SumVysq + pow(VyNew, 2);
    }
}

// caculate the kinetic energy if necessary
SumVsq = SumVxsq + SumVysq;
//      KE = 0.5 * m * SumVsq;

return;
} // end of verlet

```

```

//*****
// Temp.cc
//      Purpose: This program allows the user to scale the

```

```

//          velocities by using the desired temperature.
//*****
void Temp( Velocity Vx, Velocity Vy, double Ta )
{
    double r = sqrt(Td/Ta);
    for(int i = 1; i < n1-1; i++)
        for(int j= 1; j < n2-1; j++)
            //    for(int k = 1; k < 2; k++) // (3D)
            {
                Vx[i][j] = r * Vx[i][j];
                Vy[i][j] = r * Vy[i][j];
            }

    return;
} // end of Temp

```

```

//*****
// void Report(Position Rx, Position Ry, Position Rz,
//          Velocity Vx, Velocity Vy, Velocity Vz, int Num)
// Purpose:  reports the new positions and velocities at a
//          certain interval.
// Output:   (to Report.out) the new positions & velocities

```

```

//*****
void Report(ofstream& outFile2, ofstream& outFile3,
           ofstream& outFile4, ofstream& outFile6,
           ofstream& outFile7, ofstream& outFile9,
           ofstream& outFile10, Position Rx,
           Position Ry, long double PE)
{
    outFile2 << "                " << PE << endl;
    outFile3 << "                " << Rx[Cx][Cy] << endl;
    outFile4 << "                " << Ry[Cy][Cy] << endl;

    outFile6 << "                " << Rx[Fx][Fy] << endl;
    outFile7 << "                " << Ry[Fx][Fy] << endl;

    outFile9 << "                " << Rx[Sx][Sy] << endl;
    outFile10 << "                " << Ry[Sx][Sy] << endl;
    return;
} // end of Report

```

Bibliography

1. Alder, B.J., and Wainwright, T.E. 1957. Phase transition for a hard sphere system. *J. Chem. Phys.* **27**: 1208-1209.
2. Alder, B.J., and Wainwright, T.E. 1959. Studies in molecular dynamics. I. General method. *J. Chem. Phys.* **31**: 459-466.
3. Allen, M.P., and Tildesley, D.J. 1989. *Computer Simulation of Liquids*. Clarendon Press, Oxford.
4. Alper, K.O., Singla, M., Stone, J.L., and Bagdassarian, C.K. 2001. Correlated conformational fluctuations during enzymatic catalysis: Implications for catalytic rate enhancement. *Protein Sci.* **10**: 1319-1330.
5. Bahar, I., and Jernigan, R.L. 1999. Cooperative fluctuations and subunit communication in tryptophan synthase. *Biochemistry* **38**: 3478-3490.
6. Balabin, I.A., and Onuchic, J.N. 2000. Dynamically controlled protein tunneling paths in photosynthetic reaction centers. *Science* **290**: 114-117.

7. Cameron, C.E., and Benkovic, S.J. 1997. Evidence for a functional role of the dynamics of Glycine-121 of *Escherichia coli* dihydrofolate reductase obtained from kinetic analysis of a site-directed mutant. *Biochemistry* **36**: 15792-15800.
8. Frenkel D., and Smit, B. 1996. *Understanding Molecular Simulation*. Academic Press, San Diego.
9. Greenwald, J., Le, V., Butler, S.L., Bushman, F.D., and Choe, S. 1999. The mobility of an HIV-1 Integrase active site loop is correlated with catalytic activity. *Biochemistry* **28**: 8892-8898.
10. Hadfield, A.T., and Mulholland, A.J. 1999. Active-site dynamics of ASADH---a bacterial biosynthetic enzyme. *Int. J. Quant. Chem.* **73**: 137-146.
11. Haile, J.M. 1992. *Molecular Dynamics Simulation*. John Wiley & Sons, New York.
12. Hill, T.L. 1986. *An Introduction to Statistical Thermodynamics*. Dover Publications, Inc., New York.

13. Hoang, T.X., and Cieplak, M. 1998. Protein folding and models of dynamics on the lattice. *J. Chem. Phys.* **109**: 9192–9196.
14. Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.
15. Jencks, W.P. 1975. Binding energy, specificity, and enzymic catalysis: the circe effect. *Adv. Enzymol. Relat. Areas Mol. Biol.* **43**: 219–310.
16. Karplus, M., and Petsko, G.A. 1990. Molecular dynamics simulations in biology. *Nature* **347**: 631–639.
17. Klimov, D.K., and Thirumalai, D. 1998. Linking rates of folding in lattice models of proteins with underlying thermodynamic characteristics. *J. Chem. Phys.* **109**: 4119–4125.
18. Kohen, A., Cannio, R., Bartolucci, S., and Klinman, J.P. 1999. Enzymatic dynamics and hydrogen tunneling in a thermophilic alcohol dehydrogenase. *Nature* **399**: 496–499.
19. Loh, A.P., Guo, W., Nicholson, L.K., and Oswald, R.E. 1999. Backbone dynamics of inactive, active, and effector-bound

- Cdc42Hs from measurements of ^{15}N relaxation parameters at multiple field strengths. *Biochemistry* **38**: 12547–12557.
20. Miller, G.P., and Benkovic, S.J. 1998. Deletion of a highly motional residue affects formation of the Michaelis complex for *Escherichia coli* dihydrofolate reductase. *Biochemistry* **37**: 6327–6335.
21. Miller, G.P., and Benkovic, S.J. 1998. Strength of an interloop hydrogen bond determines the kinetic pathway in catalysis by *Escherichia coli* dihydrofolate reductase. *Biochemistry* **37**: 6336–6347.
22. Ota, N., and Agard, D.A. 2001. Enzyme specificity under dynamic control II: Principal component analysis of α -lytic protease using global and local solvent boundary conditions. *Protein Sci.* **10**: 1403–1414.
23. Pauling, L. 1948. Nature of forces between large molecules of biological interest. *Nature* **161**: 707–709.
24. Pham, D.T., and Karaboga, D. 2000. *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search,*

Simulated Annealing and Neural Networks. Springer-Verlag
Berlin Heidelberg, New York.

25. Radkiewicz, L., and Brooks, C.L. III. 2000. Protein dynamics in enzymatic catalysis: Exploration of dihydrofolate reductase. *J. Am. Chem. Soc.* **122**: 225-231.
26. Singla, M. and Bagdassarian, C.K. 2001. Dynamical tuning of enzymatic conformational fluctuations for maximal catalytic velocity. *Protein Sci.* submitted.
27. Socci, N.D., Onuchic, J.N., and Wolynes, P.G. 1999. Stretching lattice models of protein folding. *PNAS* **96**: 2031-2035.
28. Wang, F., Li, W., Emmett, M.R., Hendrickson, C.L., Marshall, A.G., Zhang, Y.-L., Wu, L., and Zhang, Z.-Y. 1998. Conformational and dynamic changes of *Yersinia* protein tyrosine phosphatase induced by ligand binding and active site mutation and revealed by H/D exchange and electrospray ionization Fourier transform ion cyclotron resonance mass spectroscopy. *Biochemistry* **37**: 15289-15299.

29. Wang, T., Miller, J., Wingreen, N.S., Tang, C., and Drill, K.A. 2000. Symmetry and design ability for lattice protein models. *J. Phys. Chem.* **113**: 8329–8336.
30. Williams, P.D., Pollock, D.D., and Goldstein, R.A. 2001. Evolution of functionality in lattice proteins. *J. Mol. Graphics Mod.* **19**: 150–156.
31. Zajicek, J., Chang, Y., and Castellino, F.J. 2000. The effects of ligand binding on the backbone dynamics of the kringle 1 domain of human plasminogen. *J. Mol. Biol.* **301**: 333–347.

VITA

Shiying Shang

Born in Beijing, China, December 28, 1977. Graduated from No. 14 high school in that city, June 1996. Earned a Bachelor of Science with a concentration in chemistry from Peking (Beijing) University, June 2000. Master of Arts candidate, the College of William and Mary, August 2000 – May 2002. The course requirements and thesis have been completed.