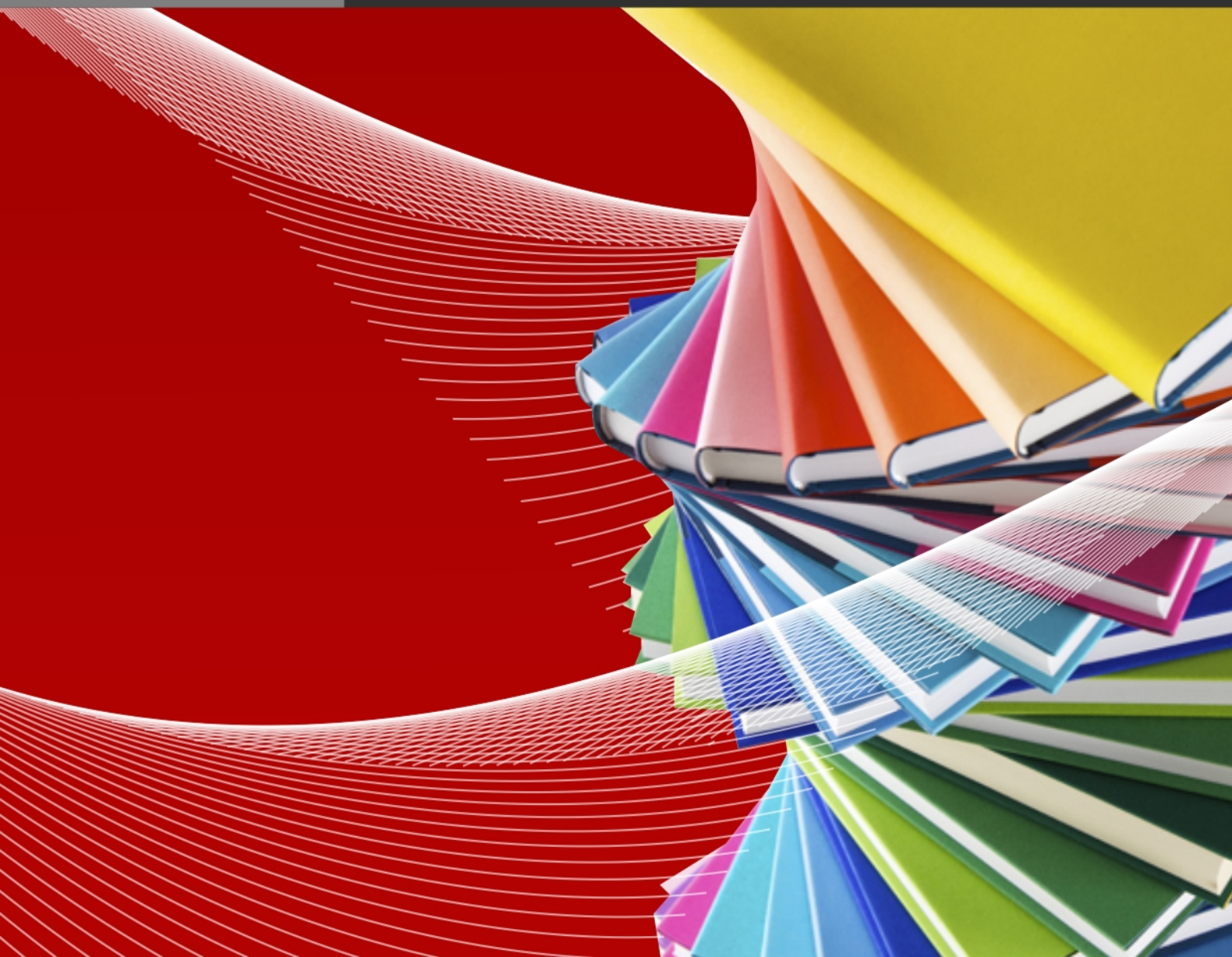
澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

Outstanding Academic Papers by Students
學生優秀作品



University of Macau

Faculty of Science and Technology

澳門大學

UNIVERSIDADE DE MACAU

UNIVERSITY OF MACAU

Improving Protein-Ligand Docking by Particle Swarm Optimization

NG CHON KIT, Marcus

Student No: D-B0-2848-7

Final Project Report submitted in partial fulfilment
of the requirements of the Degree of
Bachelor of Science in Software Engineering

Project Supervisor

Dr. Shirley W. I. Siu

08 October 2014

DECLARATION

I sincerely declare that:

1. I am the sole author of this report,
2. All the information contained in this report is certain and correct to the best of my knowledge,
3. I declare that the thesis here submitted is original except for the source materials explicitly acknowledged and that this thesis or parts of this thesis have not been previously submitted for the same degree or for a different degree, and
4. I also acknowledge that I am aware of the Rules on Handling Student Academic Dishonesty and the Regulations of the Student Discipline of the University of Macau.

Signature : _____

Name : NG CHON KIT, Marcus

Student No. : D-B0-2848-7

Date : 08 October 2014

ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisor Dr. Shirley W. I. Siu. During the progress of the project, for me this is a complete new and challenge topic. Dr. Shirley gave a step-by-step guidance and brought me a most bioinformatics knowledge. I always can be inspired from our fruitful discussions. Besides the knowledge related to this project, she also helped for proof reading of this report. I am very grateful for all the supports from Dr. Shirley. I am also grateful for Mr William helped to maintain and provide the testing machine, Dr. Ryan L. H. U for coordinated the Final Year Project. In addition, I am grateful for University of Macau gives me an opportunity to have education here and such a chance to contribute a little improvement in related area.

ABSTRACT

Molecular docking is an indispensable tool in computer-aided drug design. Given a target protein related to the disease of interest, molecular docking helps to decide if a small molecule called ligand is able to bind to the protein's binding pocket with certain level of affinity. Protein-ligand docking consists of two main steps: A conformation (a possible configuration of the molecule structure) is generated by conformational search algorithm and then the conformation is evaluated by a fitness function. The conformation which has the best value returned by the fitness function will be determined as the predicted structure. To be able to search quickly and intelligently over the huge conformational space is necessary in the virtual screening step of drug design, where millions of ligands in a drug compound library will have to be screened through by docking. In recent years, swarm intelligence algorithms have emerged as a fast and reasonably accurate technique in solving complex search problems in computer science. But the applicability of this technique has not been fully explored in the molecular docking problem.

Therefore, in this project one of the most popular swarm intelligence algorithms, particle swarm optimization (PSO) was studied for their applicability to the ligand conformational search problem in protein-ligand docking. The algorithm is, for the first time, implemented into the popular docking program AutoDock Vina, here called PSOxVina. Using a benchmark dataset of 201 experimental protein-ligand complexes, the prediction accuracy and time efficiency for docking pose prediction were rigorously tested. Remarkably, PSOxVina completes the docking process in only 54% of the time used in AutoDock Vina, but the prediction accuracy is increased by 7% measured by the averaged RMSD of the predicted structures from experimental structures. Our work demonstrates that PSO is superior to conventional search algorithms such as Monte Carlo in molecular docking.

TABLE OF CONTENTS

LIST OF FIGURES

UNIVERSIDADE DE MACAU
仁義禮知信
澳門大學

LIST OF TABLES

CHAPTER 1. INTRODUCTION

1.1 Overview

Drug design aims to discover a new medication. Newly developed drugs help people to prevent and to recover from diseases. Drug not only cures human being even animal and plant may suffer from diseases and need medical treatment, thus drug is always in a great demand. In pharmaceutical industry, traditional drug discovery is a long-term process but it is also the most profitable business. However to find a new medicine is expensive, especially in terms of human resource, time and money. Normally, a drug from design to deliver to patient, it takes 12 years and approximately US$359 million [1] then the scientists started to look for a cheaper solution for the development process. They tried to model the disease molecules to study their molecular properties and to design drugs which can disrupt the pathogenic activities of the molecules. With the aid of powerful computers and advanced algorithms, the time to identify candidate drug molecules in the pre-experimental screening process can be greatly reduced. This area of drug development using computational techniques is called the computer-aided drug design (CADD).

Nowadays, CADD becomes an important step in drug discovery [2]. The CADD software helps scientists to find out a few drug candidates from a million of chemical compounds. CADD can be classified into two types: (1) ligand-based and (2) structure-based. Ligand-based drug design depends on the chemical properties of ligand molecules to construct predictive models of new ligand without knowledge of the structure of the disease molecule. In contrast, structure-based drug design relies on the three dimensional structure of the target molecule. And by virtue of a search process it quickly finds out the possible docking poses of a ligand and ranks them by the strength of binding. This search process which places the ligand into the known protein structure is called docking.

The principles of docking are searching and scoring. There are many search algorithms such as Monte Carlo (MC), Simulated Annealing (SA), Genetic Algorithm (GA) and Swarm Intelligence (SI). In particular, Particle Swarm Optimization (PSO), one of the most popular SI algorithms, has gained some success in other bioinformatics tasks which need to look for optimal solutions from huge search spaces. Recently, there are also a few implementations of PSO on docking programs which show encouraging results, such as PSO@AutoDock[3] and FIPSDock[4].

Recently, a well-known docking software research group (*The Scripps Research Institute*) has released a new version of the docking software called AutdoDock Vina that has different programming structures and principles compared with the previous version AutoDock 4. Their test result showed a 29% improvement in accuracy and saving 500% of computing time [3]. It is a big contribution to the molecular docking field. Even the new version did improve on accuracy and faster, but it is still far from being perfect. As mentioned, PSO gained an improved performance in previous version of AutoDock but has not been tested in AutoDock Vina. Thereby, it gives me an opportunity to make an experimental test that is integrating PSO into Vina.

1.2 Objective

This project has two goals. Firstly, it is to establish the pipeline to automate the docking process. Before the protein and ligand pass to the docking software, a series of works have to be done manually. For example, a ligand has to remove useless atoms like water and metal ions and to add polar only hydrogens, locate and measure the binding pocket (search space) of the protein. To achieve this goal, some shell scripts and python programs will be implemented. Secondly, I aim to apply PSO on the AutoDock Vina docking program to reduce the time in searching for the best ligand pose while maintaining at least the same accuracy as the original AutoDock Vina. For this purpose, PSO will be implemented and tested with a large data set of high-resolution experimental protein-ligand structures.

1.3 System Environment

A docking process consists of many computational procedures, to provide the hardware configuration as a reference is important. Higher performance hardware you get, the shorter time you can achieve.

CPU	Intel Core i7 CPU 860@2.8GHz
RAM	4GB DDR3 1333MHz
Hard Disk	320GB SATA2 7200rpm
Graphic Card	ATI HD3450

Table 1: Hardware configuration of the working PC

My work was conducted in a desktop computer with hardware configurations shown in Table 1. Ubuntu Linux 11.04 was installed and AutoDock Vina requires installing the boost library, in my PC version 1.4.1 was installed. I compiled the source code by using GCC 4.5.2. I used PyMol 1.6 as a visualization tool to view the protein and ligand structures, this tool also allow users to do some modifications on the molecules.

CHAPTER 2. BACKGROUND

2.1 Protein Receptor and ligand

Protein is an organic compound that is the most complex chemical entities no matter in size, shape or texture. It consists of a sequence of amino acids joined hand-in-hand by peptide bonds. A protein structure can be characterized by its primary structure, secondary structure, and tertiary structure. A primary structure is a linear sequence of amino acids it starts from amino-terminal end to carboxyl-terminal end. Another distinct characteristic is secondary structure depends on hydrogen bonding. Two types of secondary structure are alpha helix and beta sheet. By virtue of the folding process, a linear chain of amino acid will be folded into a three-dimensional structure with which the protein acquires its function.

Many proteins, especially drug-targeted proteins, function as receptors. A receptor is a protein molecule which exists in the surface or inside the cell. A receptor can recognize and bind another molecule analogous to signal receiving. A molecule which can bind to a receptor is called a ligand, and the space that allow a ligand bound to the protein is called the binding pocket. When a ligand binds to the receptor, it is analogous to sending a signal to the receptor and the receptor will react on it. In pharmacology, a drug which has the same chemical properties of the natural ligand will be selected to inhibit the biological activity of the receptor, thus it is also called inhibitor.

Human Immunodeficiency Virus (HIV) Type 1 is a marked example. To cure HIV, bio-scientist tried to discover a ligand to inhibit its activity by interrupting the DNA replication process. In the HIV virus life cycle, HIV protease plays an important role. The virus relies on the HIV protease to reproduce itself. The protease cuts the virus polypeptide into pieces to create mature protein component which then starts to infect other cells. The HIV protease inhibitor inhibits the protease activity by binding to the protease's activity site and subsequently prevents the virus to reproduce itself.

2.2 Docking

Docking is a computational method used to predict the best binding conformation of a ligand in the protein's binding pocket. Figure 1 shows the pipeline of the docking procedure. In a docking program, the two important components are the scoring function and the search algorithm. A scoring function is a mathematical function to evaluate the binding free energy of a ligand binding to a protein. Because a ligand can be placed at different positions and orientations with respect to the protein, and can be adopted different torsional states which might affect the binding, the binding free energy would be changed according to these conformational states dramatically. It is difficult to find the globally optimal binding conformation from all possible conformational states because the entire search space is huge. Thus an efficient searching algorithm takes the responsibility to determine whether an optimal solution is reached. These two components rely on each other: If a scoring function is not accurate enough no matter how efficient the search algorithm works, it is hard to find the lowest energy conformation of the ligand at the protein pocket. On the other hand, if the scoring function is accurate but the search algorithm is not fast enough to find

the best solution because the searching is like looking a needle in a haystack then it is still difficult to find the best ligand pose. Figure 2 shows a HIV1 protein and ligand before and after docking, the bottom image shows the ligand binds to the protein with the best pose.

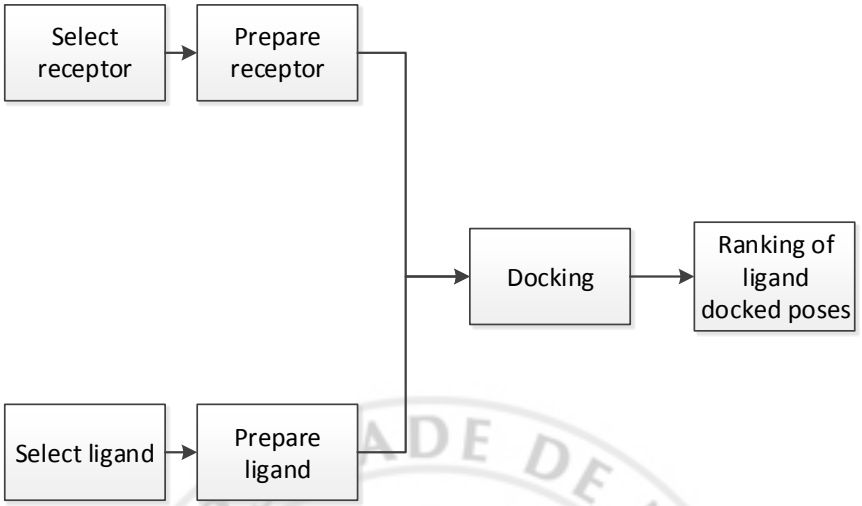


Figure 1 Flowchart of performing docking study

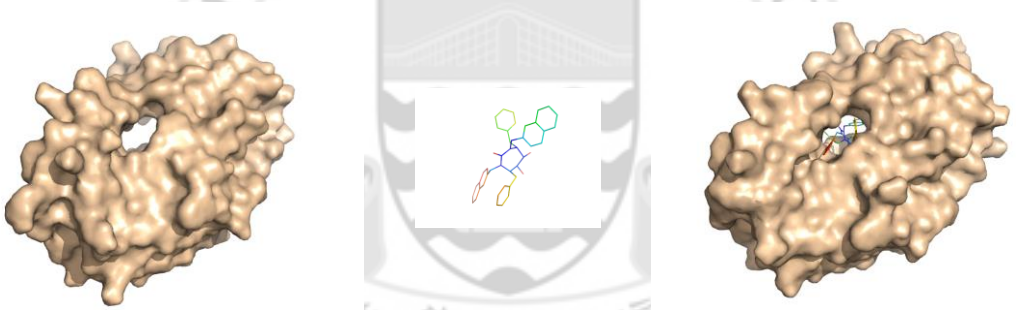


Figure 2: The Human Immunodeficiency Virus Type 1 protein (left); a ligand molecule (middle); the ligand-bound complex (right).

2.3 Search Algorithms in Docking

As the search algorithm is important, apply a suitable algorithm is essential. An ideal search algorithm should be able to find the most energetically favourable conformation (global minimum) in the search space. There are two approaches in searching: (1) a search on the entire solution space systematically, and (2) a guided search exploring part of the solution space. The former is a comprehensive search examining all possible translation, orientation, torsional states of a ligand in the protein pocket, whereas the latter applies changes (with some randomness) in the ligand conformation intelligently followed by local optimization. Popular guided search algorithms include Monte Carlo, simulated annealing and evolutionary algorithms such as genetic algorithms and swarm algorithms. In the following sections, we will introduce two evolutionary algorithms: genetic algorithm and particle swarm optimization.

2.3.1 Genetic Algorithms

Genetic algorithms were formally introduced in the United States in the 1970s by John Holland at University of Michigan [5]. This algorithm is trying to simulate the creature evolution and rules by using such as gene, chromosome, mutation, crossover, applying these principles to global search and optimization. In genetic algorithms, the solutions of the problem are called individuals some may call them as a genomes. The solutions may consist of a series of properties and evolves like a natural populations which represent in digit in the memory over time. A particle way of genetic algorithms is initial populations of individuals and begin evolve over iterative process, each iterative is called generation. In each generation, every individuals will be evaluated by a fitness function, generally the fitness function is the objective function itself. The individuals will be ranked by their fitness value compare with the previous generation. The best one will be selected to the next generation. The algorithms will be terminated at the maximum number of generations or an acceptable fitness value has been reached.

2.3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is in reference to James Kenndy and Russell Eberhart paper and was trying to simulate the social behaviour of birds flock or fish school [6]. In PSO, the flock of birds in looking for food is simulated. When one of the birds finds food at its current location, it then informs the others and all the birds are trying to move close to that location. The best solution to this problem is the food location. Each bird is called a particle defined by a vector of variables analogous to the bird's current location. A number of particles are used to represent a bird flock and each of the particles is a candidate solution. The entire area that the birds searching for food is the search space which is a combination of all possible values of all variables allowed in the vector for a particle. Imagine a bird is flying on the sky, a bird has it owns experience and able to use it to determine the flying direction and velocity. In PSO, a flock of birds flying at the same time. Each single bird can fly based on its own experience on the possible location of food (the so-called personal-best position). Its search strategy will also be affected by knowledge of all other birds' experiences (the so-called global-best position). From these two pieces of information, the bird will decide the direction and speed of the movement.

```
Define the number of particles
Initialize the position and the velocity of each particle
Loop until stopping condition is met
        For each particle
                Evaluate fitness function based on the particle position
                If fitness value better than the personal best value
                        update the personal best position to the current position
                        If fitness value better than global best value
                                update the global best position to the current position
                        End if
                End if

                Update the velocity (use Equation 1)
                Compute the new position from the new velocity (use Equation 2)

        End for
End Loop
```

Figure 3: Pseudo code of the standard PSO algorithm.

$$v_i(t+1) = w*v_i(t) + c1*rand() * (pbest_i(t) - x_i(t)) \qquad \textbf{Equation 1}$$

$$+ c2 * rand() * (gbest(t) - x_i(t))$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \qquad \textit{Equation 2}$$

where $x_i(t)$ is the position of the particle i in the t^{th} iteration, and $v_i(t)$ is the velocity of the particle i in the t^{th} iteration. Variable *pbest* and *gbest* are the personal-best and the global-best values.

Figure 3 shows the process of PSO in each iteration and the equations for position-update and velocity update are shown in Equation 1 and Equation 2. An objective function is required to evaluate the fitness of each move. In addition, a particle decides to move to the next step, a randomness factor is being considered to prevent local minimum trap. There are two parameters used to derive the velocity in the next iteration (see Equation 1). The *c1* and *c2* parameters are known as the acceleration or the learning coefficients. They determine the weight of personal best and global best. Function *rand()* generates a random number and *w* is the inertia weight. If *w=0* the new velocity is only determined by the personal and global best positions, each step of movement will be smaller. In other words, it will become more like a local search. When the value of *w* is larger, the search will become a global search.

Some papers specify a communication topology between the birds. They may enclose in ring, square, etc. but their concepts are the same, i.e. to gather information from others.

2.4 Scoring Function

As mentioned, scoring function is one of the critical components of docking. A perfect scoring function can evaluate the interaction between protein and ligand rapidly and accurately. There are three applications using scoring function in protein-ligand docking [7]: (1) Determine the binding site and the binding mode, (2) calculate the binding affinity, (3) for virtual screening. The second one should be focused because in this project the scoring function is used to calculate the binding affinity of the protein-ligand interaction. In AutoDock Vina, the scoring function was inspired by the knowledge-based X-score function and parameterized using the PDBbind protein-ligand complexes data set. It includes physical atomic interactions such as van der Waals interactions and electrostatic interactions (Coulombic interaction), and bond rotations. The general form of the AutoDock Vina scoring function is:

$$C = \sum_{i<j} f t_i t_j (r_{ij}) \qquad \textit{Equation 3}$$

where i stands for a protein atom, j stands for a ligand atom. r_{ij} is the distance between them. For each kind of atom, assigned a type t, and a symmetric set of interaction function f_{titj} of the protein-ligand atom distance. The interaction function is defined by

$$f_{titj}(r_{ij}) \equiv h_{titj}(d_{ij}) \qquad \textit{Equation 4}$$

where $d_{ij} = r_{ij} - R_{ti} - R_{tj}$, where R_t is the van der Waals radius of atom type t and h_{titj} is the weighted sum of five different types of interactions between all protein and ligand atom pairs, including three different types of hydrophobic interactions and hydrogen bonding interactions.

2.5 Docking with AutoDock

AutoDock is a computational docking software set, it is used to predict molecular, and bind to a receptor of known 3D structure and wildly use in drug design. This software set was developed by The Scripps Research Institute. TSRI locates in San Diego, California, is one of the most well-known research and education in biomedical sciences and the largest non-profit, private biomedical research organization. Up to date, there are two current distributions of the AutoDock docking programs: AutoDock 4 and AutoDock Vina. AutoDock 4 is under GNU General Public License and Vina is under Apache License. Both of them are free to download and open source. AutoDock Vina is a brand new generation, TSRI re-designed the whole programe even for the searching algorithm and scoring function. AutoDock 4 uses Lamarckian Genetic Algorithm for the lowest energy conformation search only and Vina uses Monte Carlo to perform global searching as a global minimum is the lowest energy value of a collection of local minima, to perform the local search, Vina uses Broyden-Fletcher-Goldfarb-Shanno (BFGS) as an algorithm for nonlinear unconstrained optimization [3]. The new version was proved to be successful, generally it improves the accuracy. From Figure 4, the chart shows that the 190 docking complex which RMSD<2 Å increase 29% compare with AutoDock 4 Vina also allows multithreading on multicore machine, so it speedup the docking process significantly compared with AutoDock4, From Figure 5, a single threaded Vina runs 62 times faster than AutoDock4, a multithreaded Vina runs 7.25 times faster when using 8 cores on the testing machine.

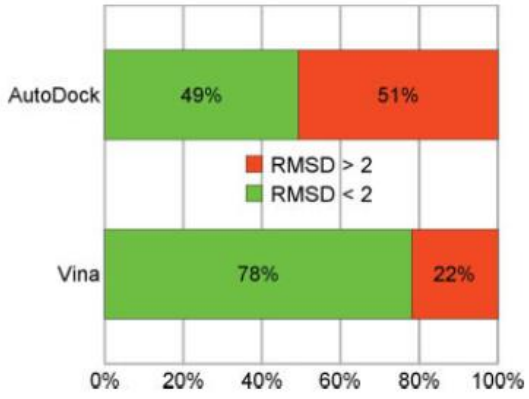


Figure 4: The fraction of the 190 test complexes for which RMSD < 2Å was achieved by AutoDock and Vina [3]**.**

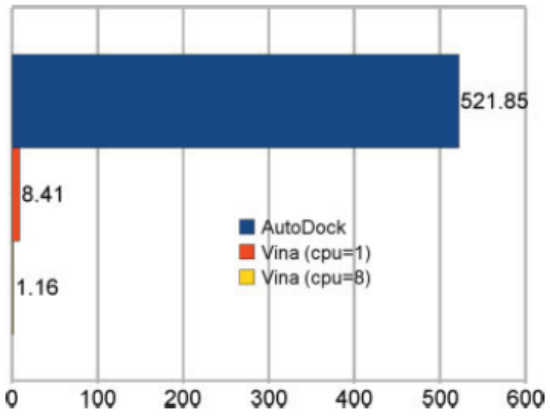


Figure 5: Average time, in minutes per complex, taken by AutoDock, single-threaded Vina and Vina with eight-way multithreading. Machines with two quad-core 2.66GHz Xeon processors were used in the experiment. [3]

There are numerous AutoDock-derived docking programs with improved accuracies and computational efficiencies. Regarding the latter, there is a recently developed Vina-based docking program called QuickVina. The authors of QuickVina found that Vina uses Monte Carlo as the global search method. The Monte Carlo consists of random mutation and local search uses Quasi-Newton method together with BFGS. The authors stated that the highest computational cost is the local search, BFGS – an approximate method to Newton method. Calculating first and second order of derivatives, it is the most expensive calculation within the Monte Carlo. QuickVina trying to reduce the computational cost of local search, they stated that searching a global minimum means looking a stationary point where its *gradient=0*. By the sides of the stationary point, the curve may be decreasing or increasing where its derivative sign is negative or positive. By this principle, a database have to store all the points that have been reached by local search before and if the current point is opposite sign with its stored neighbor points then local search of the current point can be performed. The test result shows reduce the running time of docking from 6,640 seconds—roughly 1 hour and 50 minutes—to merely 263 seconds, which is 4 minutes 23 seconds [8].

2.6 Application of PSO on the docking problem

As mentioned previously, the use of PSO on the docking program has not yet been fully explored. Until now, there were only two studies implementing PSO on docking programs. They are PSO@AutoDock and FIPSDock.

2.6.1 PSO@AutoDock

PSO@AutoDock is a variant version of AutoDock 3, a previous version of AutoDock 4. It was developed by Institute of Biochemistry, University of Leipzig. The research group saw the shortage of basic PSO, they changed the velocity formula, improve the way that the particles updated velocity and using local search strategy then reproduced based on PSO, they called varCPSO and varCPSO-ls [3]. varPSO is a short form of velocity adaptive and regenerative CPSO which is an extension of the traditional Constricted Particle Swarm Optimization together with the velocity update which is only update the velocity when the current iteration fitness value is worse than the previous one. The advantage of varPSO is speed-up the particles move in the

search space and reduce the computational cost. varCPSO-ls is based on varPSO and involved local search strategy. For each iteration, the original PSO will calculate the velocity according to the global best and personal best. However, in this algorithm each iteration the lowest binding energy particle will be considered as swarm best. The swarm best will be used to instead of global and personal best as well as this particle will be flag marked and going for local search by maximal 50 optimization steps [3]. Other particles will move according to their experiences (global best and personal best).

2.6.2 FIPSDock

FIPSDock is a variant algorithm of PSO introduced by Y. Liu et al. FIPS stands for fully informed particle swarm. In FIPS the particles communication and the source of information to adjust the velocity are different from the original PSO. FIPS states that it is essential to have a suitable topology for the particles changing information. Hence, FIPS uses URing as the topology to solve the complex conformational sampling problem. In URing topology, all the particles are connected to its neighbours and update the velocity based on the neighbour information. Later FIPS was applied to AutoDock 4 by using the scoring function which was defined by default, the results were compared with published results for a few state-of-the-art docking programs including PSO@AutoDock, SODOCK, AutoDock, Glide, GOLD, FlexX, Sur- flex, and MolDock. FIPSDock has obtained a successful predicting rate of 93.5% (successful docking run with RMSD < 2Å) for 77 complexes and outperformed all other docking programs [4].

CHAPTER 3. Design

In this project, our goal is to investigate the applicability of PSO on the protein-ligand docking problem in terms of time efficiency and prediction accuracy. To this end, a reliable data set of protein-ligand complexes is required as a measurement of performances. The PSO algorithm is designed and implemented into the AutoDock Vina replacing the original Monte Carlo search algorithm. In addition, in order to conduct the tests efficiently, an automated pipeline is implemented to pre-process the structural data before feeding them to the docking program. Results will be collected and assessed in an automated fashion. In this chapter, we will present the details of the test data, the design of the automated docking pipeline and the data structure design of the PSO implementation for optimal ligand-pose search.

3.1 The Data

The PDBbind database used in this study was released in 2012. It contains 201 protein-ligand complexes as well as the binding affinity data of those complexes. The PDBbind database was originally developed by Prof. Shaomeng Wang's group (http://sw16.im.med.umich.edu) at the University of Michigan in USA. There are two distributions, one is called the “core set” and another one is called the “refined set”. These two sets were filtered from 87,085 proteins which determined the structures by experiment. The refined set selected protein-ligand complexes from the 87,085 proteins with better quality and need to match several filters regarding the binding data. The core set data is selected from the refined set as follows: For each family which contains at least five members, the highest, the lowest and the medium binding structures were selected as the representatives of the family [9]. Because the data set contains representative structures and diversely covering all known protein-ligand complex families, the core set is suitable for validating prediction models. In my project the core set will be used.

Each complex in the core set, named with its PDB ID contains a structure of the protein, the ligand, as well as the binding pocket. There are in total 201 complexes in the core set. As shown in Figure 6, there are mostly small ligands, with the number of rotatable bonds (torsion size) between 0 and 10. There are relatively few mid-size ligands, and rarely ligands with torsion sizes > 16.

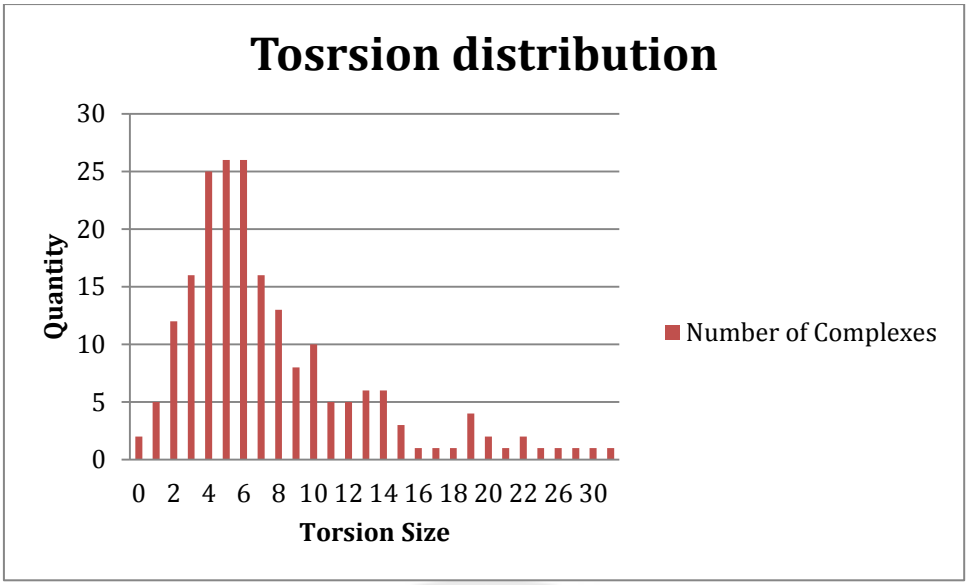


Figure 6: Distribution of ligand's torsion size in the PDBbind core set.

3.2 Pipeline Automation

Figure 1 and Figure 8 presents the pipeline of docking. Before I implemented the PSO in AutoDock Vina, I would like establish an automated script to select molecules and calculate the binding pocket size.

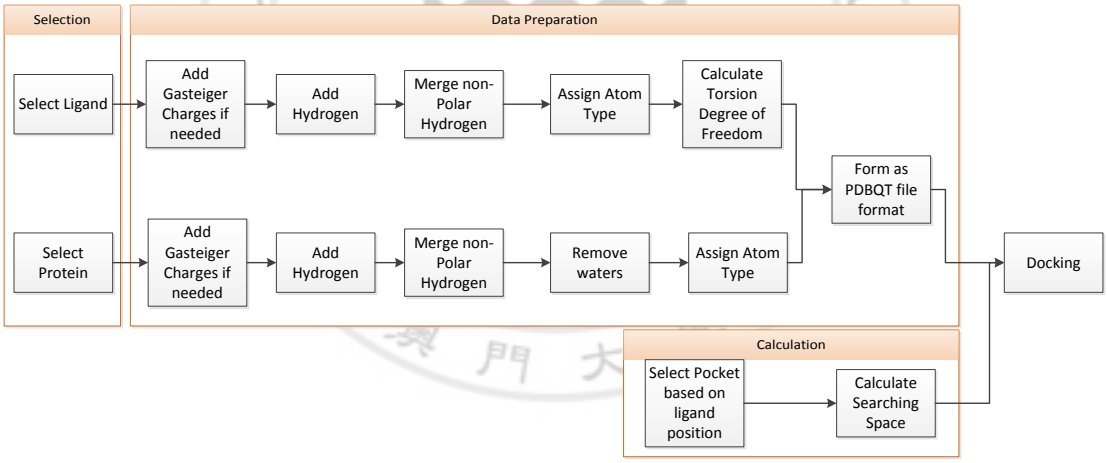


Figure 7 Pipeline of the automated pre-docking process

```
For each protein-ligand complex
        Select ligand
        Add Gasteiger charges, add hydrogen, merge non-polar hydrogen, assign atom type
and calculation torsion degree of freedom
        Generate modified ligand in PDBQT file format
        Select protein
        Add Gasteiger charges, add hydrogen, merge non-polar hydrogen, remove waters
and assign atom type
        Select ligand (actually you derive the pocket position by the ligand position,
and not directly selecting the pocket)
        Estimate the size of the binding pocket
        Randomize the ligand structure
        Generate docking configuration file
        Docking with Vina
End For
```

Figure 8 Pseudo code of the automated pipeline

3.2.1 Selection

Given the PDB ID of a protein-ligand complex, the script itself should be able to select the suitable receptor with the corresponding ligand. According to my test data set, for each protein, the data set gives a protein structure, protein binding pocket which contains a bunch of atoms that around the binding pocket and a bound ligand structure.

3.2.2 Structure Files Preparation

One of the most common file formats for 3D molecular structures is the Protein Data Bank (PDB) format. However, in AutoDock Vina it only supports PDBQT with 'Q' an addition of Gasteiger charges and 'T' for AutoDock atom type. In addition, PDB file has some problems of missing atom, added waters, more than one molecule, chain breaks, alternate locations, etc. [10]

From the AutoDock ligand and receptor tutorial [10], a ligand before docking should be done with the following operations: (1) If no partial charge or if each of the charges is zero, Gasteiger charges should be added for the entire ligand, (2) add hydrogen atoms, (3) merge non-polar hydrogen, (4) assign AutoDock atom type to each atom, and (5) calculate the torsion degree of freedom, then the ligand PDBQT file is ready. For receptor structure, it should be done with the following: (1) If no partial charge or if each of the charges is zero, Gasteiger charges should be added to each atom, (2) add hydrogens, (3) merge the non-polar hydrogens, (4) remove waters, and (5) assign AutoDock atom type to each atom then the receptor PDBQT file is ready.

3.2.3 Search Space Calculation

Before passing the receptor and ligand files to the docking program, we have to specify the search space. The search space should cover the whole binding pocket. If defined search space can perfectly fit to the pocket, it helps to save the docking time and more accurate for docking because the solution space is smaller. Hence the pocket structure with each atom 3D-coordinates is given where the search space centre can be calculated by:

$$\begin{cases} Xc = \dfrac{\sum_{i=0}^{n} x_i}{n} \\ Yc = \dfrac{\sum_{i=0}^{n} y_i}{n} \\ Zc = \dfrac{\sum_{i=0}^{n} z_i}{n} \end{cases}$$

Equation 5

where n is the number of atoms in the binding pocket. And the length of the each dimension can be calculated in following equation:

$$\begin{cases} Lx = X_{max} - X_{min} \\ Ly = Y_{max} - Y_{min} \\ Lz = Z_{max} - Z_{min} \end{cases}$$ ***Equation 6***

Where the subscript *max* and *min* represent the largest and smallest coordinates found for each dimension.

3.3 Particle Swarm Optimization

In the previous chapter, PSO algorithm has been shown in pseudo code (Figure 3). By the concept of PSO, such as a flock of birds, bird movement and movement factor, choosing a suitable architecture can make the logic and coding more clear. In Vina, C++ programming language has been used. C++ provides object-oriented programming, it is easy to implement PSO on C++ by using the idea of object with associate methods.

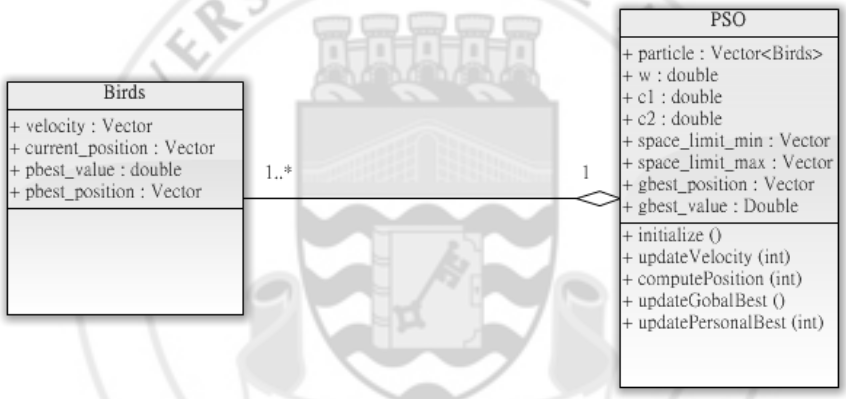


Figure 9 Abstract Class Diagram of PSO

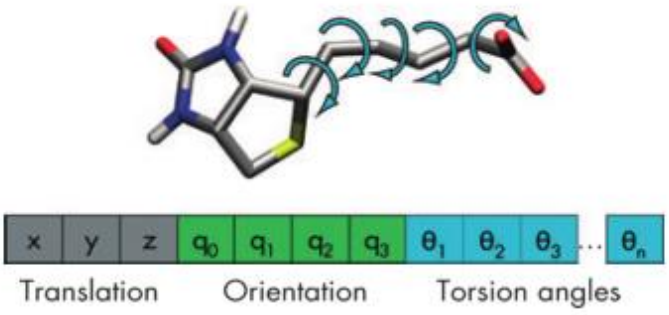


Figure 10 Vector representation of the ligand conformation and its position at the binding pocket of the protein [3]***. This vector is the position vector of each bird in PSO.***

Figure 9 shows the primary class diagram of PSO. The conformation of a ligand can be represented by the selected torsional states, its position and orientation with respect to the protein structure. Therefore, each bird should encode the ligand conformation with a vector of $7+N_{tors}$ variables as shown in Figure 10: the ligand's centre position consisting of the x, y, z coordinates; its orientation encoded using a quaternion (q_0, q_1,

$q_2, q_3)$ where (q_0, q_1, q_2) defines a vector originated at the centre of the ligand and q3 is the self-rotation of the ligand along this vector; and a torsion angle θ_i for each rotatable bond in the ligand. Each degree of freedom represents a dimension in the search space where the birds will explore over it. In a standard PSO, all the birds will firstly be initialized its position vector $x_i(t=0)$ as well as the velocity vector $v_i(t=0)$ within the search space randomly. Then they start to move according to the position update (Equation 1) and velocity update equations (Equation 2). If a bird moves outside of the search space, it will be put back to the search space to a random position. The search step will iterate until the stopping criteria is met. Figure 11 shows the PSO flowchart for protein-ligand docking.

UNIVERSIDADE DE MACAU
仁義禮知信
澳門大學

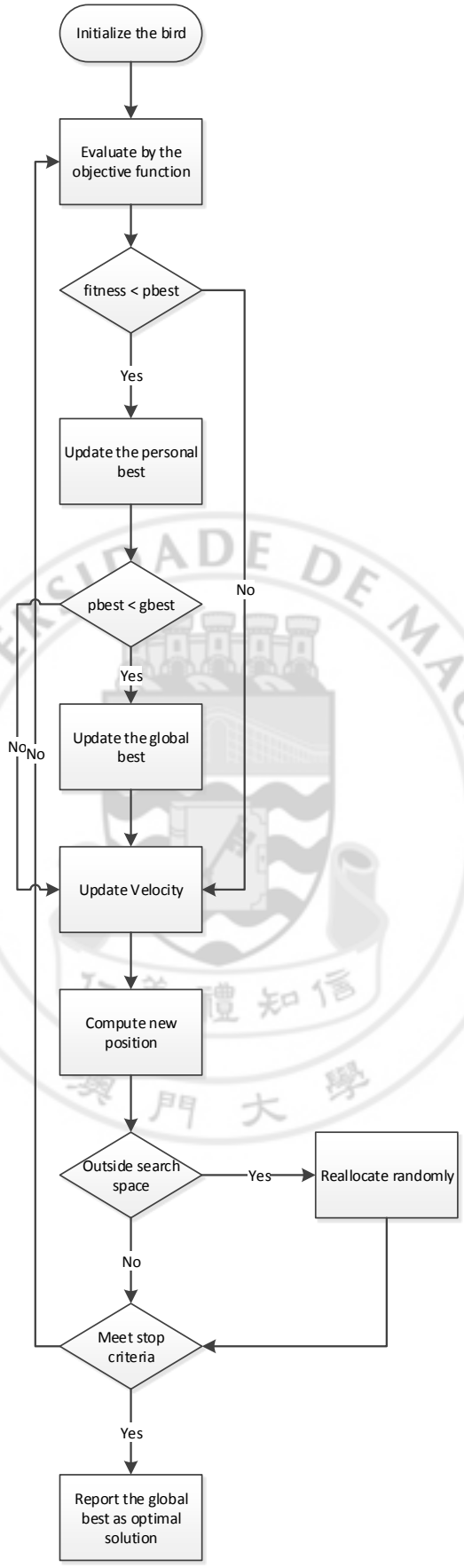


Figure 11 Flowchart of the designed PSO

3.3.1 Stopping Criterion

A stopping criterion I think is worth to discuss about. Let say those birds can fly infinitely. By the time, all the birds may converge to the same point which is the best well-known solution for them. However, if a stopping criterion is not well designed, those birds may keep flying near the best point, but may improve a very little or even haven't improved the solution but keep spending time on it. And my project goal is to speed up the docking process, it is necessary to avoid the useless calculation. To design a stopping criterion before testing is difficult, I have to observe the regular converge steps then design within how many steps if there is a very little bit change or even no change to decide when is the proper time to terminate the loop. The more detail of stopping criteria will be discussed in CHAPTER 6.

UNIVERSIDADE DE MACAU
仁義禮知信
澳門大學

CHAPTER 4. Implementation

4.1 Pipeline Automation

In CHAPTER 3, an automated procedure has been defined (Figure 7) as well as the pseudo code. To achieve the routine that I defined, Shell Script is an easy approach for implement such kind of simple routine. AutoDock Tools provides a set of python program, those python programs have been already implement receptor and ligand operation such as add the charges, remove water and assign AutoDock atom type etc. In the shell script just simple to call the python program then the data preparation part can be processed by the python programs. A PDB file actually is a text file, all the atoms properties are written in plain text. An easy way for calculating the binding pocket size and position, to use AWK program, a language for processing text files.

4.1.1 Selection

From Figure 8, a complex selection is simply to pick up the protein-ligand by their name. The most common naming rule is ${PDB_CODE}_${molecule_type}.${file_type}. I can summarize the filename in a text file, the Shell Script can simply fetch the file to access the molecule structure.

```
infile=index_of_the_comlexes
for nameList in `grep -v "\#" $infile | cut -f1 -d\ ` ; do
…
…
done
```

Figure 12: Partial shell script code of fetching each molecule complex

Figure 12 shows the main loop of fetching the molecule complex from the PDB code list.

4.1.2 Structure Files Preparation

As the pipeline outline, the preparation consists the following operations:

(1) If no partial charge or if each of the charges is zero, Gasteiger charges should be added

(2) Add hydrogen atoms

(3) Merge non-polar hydrogen

(4) Assign AutoDock atom type to each atom / Remove waters

(5) Calculate the torsion degree of freedom

There are two python programs provided by AutoDock Tools, they are prepare_receptor4.py and prepare_ligand4.py. These two programs provide a list of arguments which can perform my designed operations, the lists are shown in Figure 13 and Figure 14.

```
Usage: prepare_ligand4.py -l filename

    Description of command...
         -l     ligand_filename (.pdb or .mol2 or .pdbq format)
    Optional parameters:
        [-v]    verbose output
        [-o pdbqt_filename] (default output filename is ligand_filename_stem + .pdbqt)
        [-d]    dictionary to write types list and number of active torsions
        [-A]    type(s) of repairs to make:
                 bonds_hydrogens, bonds, hydrogens (default is to do no repairs)
        [-C]    do not add charges (default is to add gasteiger charges)
        [-p]    preserve input charges on atom type, eg -p Zn
               (default is not to preserve charges on any specific atom type)
        [-U]    cleanup type:
                 nphs_lps, nphs, lps, '' (default is 'nphs_lps')
        [-B]    type(s) of bonds to allow to rotate
               (default sets 'backbone' rotatable and 'amide' + 'guanidinium' non-rotatable)
        [-R]    index for root
        [-F]    check for and use largest non-bonded fragment (default is not to do this)
        [-M]    interactive (default is automatic output)
        [-I]    string of bonds to inactivate composed of
                   of zero-based atom indices eg 5_13_2_10
                   will inactivate atoms[5]-atoms[13] bond
                                and atoms[2]-atoms[10] bond
                      (default is not to inactivate any specific bonds)
        [-Z]    inactivate all active torsions
                      (default is leave all rotatable active except amide and guanidinium)
        [-g]    attach all nonbonded fragments
        [-s]    attach all nonbonded singletons:
                   NB: sets attach all nonbonded fragments too
                      (default is not to do this)
```

Figure 13 Arguments of prepare_ligand4.py

```
Usage: prepare_receptor4.py -r filename

    Description of command...
         -r   receptor_filename
        supported file types include pdb,mol2,pdbq,pdbqs,pdbqt, possibly pqr,cif
    Optional parameters:
        [-v]  verbose output (default is minimal output)
        [-o pdbqt_filename]  (default is 'molecule_name.pdbqt')
        [-A]  type(s) of repairs to make:
             'bonds_hydrogens': build bonds and add hydrogens
             'bonds': build a single bond from each atom with no bonds to its closest neighbor
             'hydrogens': add hydrogens
             'checkhydrogens': add hydrogens only if there are none already
             'None': do not make any repairs
             (default is 'None')
        [-C]  preserve all input charges ie do not add new charges
             (default is addition of gasteiger charges)
        [-p]  preserve input charges on specific atom types, eg -p Zn -p Fe
        [-U]  cleanup type:
             'nphs': merge charges and remove non-polar hydrogens
             'lps': merge charges and remove lone pairs
             'waters': remove water residues
             'nonstdres': remove chains composed entirely of residues of
                     types other than the standard 20 amino acids
             'deleteAltB': remove XX@B atoms and rename XX@A atoms->XX
             (default is 'nphs_lps_waters_nonstdres')
        [-e]  delete every nonstd residue from any chain
              'True': any residue whose name is not in this list:
                     ['CYS','ILE','SER','VAL','GLN','LYS','ASN',
                     'PRO','THR','PHE','ALA','HIS','GLY','ASP',
                     'LEU', 'ARG', 'TRP', 'GLU', 'TYR','MET',
                     'HID', 'HSP', 'HIE', 'HIP', 'CYX', 'CSS']
              will be deleted from any chain.
              NB: there are no  nucleic acid residue names at all
              in the list and no metals.
             (default is False which means not to do this)
        [-M]  interactive
             (default is 'automatic': outputfile is written with no further user input)
        [-d dictionary_filename] file to contain receptor summary information
```

Figure 14 Arguments of prepare_receptor4.py

To perform my targeted operations, argument –A and –U, respectively to add hydrogens and merge charges, remove non-polar hydrogens, lone pairs and water. The programs will add the gasteiger charges, calculate the torsion degree of freedom and assign the AutoDock atom type automatically. The detail code is shown in Figure 14

```
prepare_receptor4.py -r {$nameList}_protein.pdb -A 'hydrogens' -o {$nameList}_protein.pdbqt -U 'nphs_lps_waters'

prepare_ligand4.py -l {$nameList}_ligand.mol2 -o {$nameList}_ligand.pdbqt -A 'hydrogens' -U 'nphs_lps_waters'
```

Figure 15 Calling python program in Shell script

4.1.3 Search Space Calculation

The calculation of search space depends on the pocket structure information. The pocket file format is PDB, the atom records is stored in plain text line by line. The atom records format is shown as Figure 16.

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	"ATOM "	
7 - 11	Integer	serial	Atom serial number.
13 - 16	Atom	name	Atom name.
17	Character	altLoc	Alternate location indicator.
18 - 20	Residue name	resName	Residue name.
22	Character	chainID	Chain identifier.
23 - 26	Integer	resSeq	Residue sequence number.
27	AChar	iCode	Code for insertion of residues.
31 - 38	Real(8.3)	x	Orthogonal coordinates for X in Angstroms.
39 - 46	Real(8.3)	y	Orthogonal coordinates for Y in Angstroms.
47 - 54	Real(8.3)	z	Orthogonal coordinates for Z in Angstroms.
55 - 60	Real(6.2)	occupancy	Occupancy.
61 - 66	Real(6.2)	tempFactor	Temperature factor.
77 - 78	LString(2)	element	Element symbol, right-justified.
79 - 80	LString(2)	charge	Charge on the atom.

Figure 16: Data fields in PDB file [11]**.**

```
REMARK  5 active torsions:
REMARK  status: ('A' for Active; 'I' for Inactive)
REMARK    1  A    between atoms: C4_1  and  O4_2
REMARK    2  A    between atoms: C3_3  and  O3_4
REMARK    3  A    between atoms: C2_5  and  O2_6
REMARK    4  A    between atoms: C5_14  and  C6_15
REMARK    5  A    between atoms: C6_15  and  O6_16
ROOT
ATOM      1  C4  KIF d   1      31.473  66.725   6.804  0.00  0.00
ATOM      2  C3  KIF d   1      32.690  66.792   7.805  0.00  0.00
ATOM      3  C2  KIF d   1      32.508  66.155   9.224  0.00  0.00
ATOM      4  C1  KIF d   1      30.936  65.555   9.352  0.00  0.00
ATOM      5  N9  KIF d   1      30.612  65.494  10.776  0.00  0.00
ATOM      6  C8  KIF d   1      29.648  66.327  11.098  0.00  0.00
ATOM      7  O8  KIF d   1      29.209  66.443  12.243  0.00  0.00
ATOM      8  C7  KIF d   1      29.230  67.057   9.838  0.00  0.00
ATOM      9  O7  KIF d   1      28.344  67.925   9.744  0.00  0.00
ATOM     10  N   KIF d   1      30.015  66.581   8.848  0.00  0.00
ATOM     11  C5  KIF d   1      30.095  67.161   7.449  0.00  0.00
ATOM     12  H8  KIF d   1      31.070  64.890  11.428  0.00  0.00
ENDROOT
BRANCH  11  13
ATOM     13  C6  KIF d   1      28.926  66.658   6.558  0.00  0.00
BRANCH  13  14
ATOM     14  O6  KIF d   1      27.646  66.911   7.160  0.00  0.00
ATOM     15  H12 KIF d   1      27.597  66.464   7.997  0.00  0.00
ENDBRANCH  13  14
ENDBRANCH  11  13
BRANCH   1  16
ATOM     16  O4  KIF d   1      31.742  67.610   5.713  0.00  0.00
ATOM     17  H2  KIF d   1      32.540  67.338   5.276  0.00  0.00
ENDBRANCH   1  16
BRANCH   2  18
ATOM     18  O3  KIF d   1      33.851  66.306   7.145  0.00  0.00
ATOM     19  H4  KIF d   1      34.029  66.844   6.383  0.00  0.00
ENDBRANCH   2  18
BRANCH   3  20
ATOM     20  O2  KIF d   1      33.247  65.323   9.678  0.00  0.00
ATOM     21  H6  KIF d   1      32.951  65.078  10.547  0.00  0.00
ENDBRANCH   3  20
TORSDOF 5
```

Figure 17 An example PDB file. Showing here is the ligand structure of 1ps3.

As the x,y,z coordinates, positions and offsets have been given. Figure 17, is a ligand data file store in PDB format. By Equation 5 and Equation 6, it is easy to calculate the search space which is defined by the binding pocket. The detail implementation code please refers to CHAPTER 10.2 Automated Pipe-line Scripts

4.2 Particle Swarm Optimization

After defined the class and some operations, a real implementation can be carried out. Figure 18 shows the pseudo code of Monte Carlo procedure of AutoDock Vina.

```
Loop until max_step

        Mutate(Ligand.vector)
        Local_Search(Ligand.vector)
        If(Metropoils_accept(Ligand.vector))
                Local_Search(Ligand.vector)
                If( score(Ligand.vector) < score(Best) )
                        Best = Ligand.vector
                End If
        End If

EndIf
End Loop
```

Figure 18 Pseudo code of Monte Carlo in Vina

To find a place for placing the PSO, I have reviewed deeper for the mutate function and local optimization function. The mutate function just give a random number for translation, orientation and rotation. The principle is to mutate the structure randomly then go for the local search after certain amount of iterations, a refined structure can be found. The most functional part is the local search therefore I decide to change the random mutation into mutate the structure by PSO. The purpose is speed-up the time, I think an intelligent algorithm rather than given a random mutation can rapidly get close to the best solution.

Therefore, the mutate function is revised by applying PSO. The PSO structures design has been done in the previous chapter. Table 2 shows the member functions of PSO and its description and parameters. The source code of these functions can be found in CHAPTER 10.3 Source Code of PSOxVina

void init(rng&,conf&)	Initialize PSO(randomize the vector, velocity, create number of birds)
void updateVelocity(rng&, int)	Calculate the new velocity of N-degree of freedom and assign to the birds.
void updateVelocityO(rng&, int)	
void updateVelocityT(rng&, int, sz)	
void computeNewPosition(int)	Compute the new pose of N-degree of freedom and assign to the birds.
void computeNewOrientation(int)	

void computeNewTorsion(rng&, int, sz)	
void updatePersonalBest(int,double)	Update the personal best fitness value
void updateGlobalBest(int)	Update the global best fitness value
void updateBestPosition(int,vec);	Set the personal best vectors to pbest variable.
void updateBestOrientation(int,qt)	
void updateBestTorsion(int,fl,sz)	
vec getCurrentPosition(int)	Return the current pose information
qt getCurrentOrientation(int)	
fl getCurrentTorsion(int,sz)	
void print();	Print all the birds information

Table 2 Member functions of the PSO class.

CHAPTER 5. Testing

5.1 Pipeline Automation

After implementing the pipeline, this chapter is to tell how to ensure the implementation is correct and achieved what I expected from the previous chapter.

Figure 7 shows the processes of pre-docking. In evaluation chapter, I will put some figures to ensure the automate process can be carried out. I expect there are 3 new files. A configuration file and two PDBQT files will be created after the automated process.

5.2 Particle Swarm Optimization

After implement the particle swarm optimization and the goal of this project is very clear. I defined several milestones or test stages for ensure PSO works fine and I am in the progress to my goal that I stated at the beginning, i.e. speed up without losing accuracy.

The following tasks will be carried out in the evaluation part.

1. **Objective**
   Apply PSO to AutoDock Vina and works effectively and maintain accuracy not worse than Vina

   Prerequisite
   Vina can be successfully compiled and run

   Input
   Prepared ligand and receptor file

   Expected Output
   Approximately accurate conformation to Vina docked conformation

   Description
   This test will mainly focus on whether PSO works correctly, will print out the information of the birds to see are those birds really ‘flying’ over the search space. Then will docking the core set using the automated script to see can the accuracy be maintained not worse than Vina. Vina docking will also be performed for comparison.

   Exception
   If the birds not ‘flying’ over the search space or the velocity doesn’t update, the debug action is required.
   If the applied PSO no matter the RMSD worse or better, the next task will be carried out.

2. Objective

Tuning parameters, including w, c1, c2 and number of birds against the docking time and accuracy.

Prerequisite

The first task has to be completed.

Input

Approximately accurate conformation

Expected Output

A set of parameters that has accurate conformation and rapidly dock

Description

This test aims to find the best combination of parameter w inertia weight, c1 coefficient of personal best, c2 coefficient of global best. Will try different combinations to find the best parameters which are rapidly and accurate. The start point will base on the previous test means the first test will use the same parameters as previous.

Exception

Keep changing the parameters until find the best combination against the accuracy and docking time.

3. Objective

Aim to find the most suitable stopping criterion and parameters by analysis the test cases.

Prerequisite

Some test cases have been performed

Input

Detail test data log from previous tests.

Expected Output

Stopping criterion is found

Description

This test will mainly analysis the previous test case data logs, to find when will the birds converge together in other words the best solution has been found**.** Then design a stopping criterion and find the converge timing point as the stopping parameter.

Exception

Applied the stopping criterion but didn't speed up. Then will keep changing the stopping parameter(s) systematically.

CHAPTER 6. Evaluation

6.1 Pipeline Automation

```
drwxr-xr-x   2 kami kami    4096 2014-05-30 18:02 .
drwxr-xr-x 205 kami kami   12288 2014-05-30 17:42 ..
-rw-r--r--   1 kami kami    3004 2014-04-28 23:00 1ps3_ligand.mol2
-rw-r--r--   1 root root    2267 2014-05-30 17:47 1ps3_ligand_out.pdbqt
-rw-r--r--   1 root root    1927 2014-05-30 17:45 1ps3_ligand.pdb
-rw-r--r--   1 root root    2200 2014-05-30 17:45 1ps3_ligand.pdbqt
-rw-r--r--   1 kami kami    2005 2014-04-28 23:00 1ps3_ligand.sdf
-rw-r--r--   1 root root    1994 2014-05-30 17:47 1ps3_ligand_vinadocked.pdb
-rw-r--r--   1 kami kami   46173 2014-04-28 23:00 1ps3_pocket.pdb
-rw-r--r--   1 kami kami 1151597 2014-04-28 23:00 1ps3_protein.pdb
-rw-r--r--   1 root root  802696 2014-05-30 17:45 1ps3_protein.pdbqt
-rw-r--r--   1 root root     104 2014-05-30 17:45 config.txt
-rw-r--r--   1 root root     521 2014-05-30 17:47 fit.log
-rw-r--r--   1 root root    3458 2014-05-19 23:03 fit.pdb
-rw-r--r--   1 root root      10 2014-05-30 17:47 rmsd2expt.log
-rw-r--r--   1 root root    1335 2014-05-30 17:47 vina.log
```

Figure 19 Complex 1ps3 files listing

Figure 19 shows the 1ps3 directory structure. Highlight with red color is the output file after docking process. Highlight with green color are the files that the automated script generated. Config.txt includes the search space size and coordinates (see Figure 20). In addition, the file format which is generated from PSOxVina is PDBQT, to calculate the accuracy, a conversion is needed. 1ps3_ligand_vinadocked.pdb is the converted file from the docking result file. Further will compare with 1ps3_ligand.pdb which is the ligand structure before docking.

```
kami@kami-OptiPlex-980:~/coreset_sh/corset_pso/1ps3$ more config.txt
center_x = 31.951
center_y = 65.5053
center_z = 7.63888
size_x = 33.452
size_y = 27.612
size_z = 35.136
```

Figure 20 1ps3 config.txt content

6.2 Particle Swarm Optimization

By achieve the tasks that I talked in the previous chapter. Different task may also spill down into different sub-tests. All the test cases will be tested on the same machine with 8 threads and 8 cores using PDBbind 2012 core set. And the number of birds represents how many birds in a single thread. Hence, totally number of birds should multiply number of threads. Table 3 and Table 4 show the averaged docking result from five separate docking runs using AutoDock Vina. Since Vina somehow will throw randomize number for calculation, to ensure the data is reliable an average result is taken. Later all the results compare with Vina refers to these two results. The

main metrics to assess the docking performance are RMSD and correlation they are calculated respectively by Equation 9 and Equation 10.

$$RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\delta_i^2} \qquad \textit{Equation 7}$$

where N is the number of pair of same position atoms in two molecules. δ is the distance between the pair of atoms.

$$Correlation(X,Y) = \frac{covariance(X,Y)}{\sigma_x \sigma_y} \qquad \textit{Equation 8}$$

where X and Y are the docked binding affinity and expected binding affinity. σ is the standard deviations.

Test case #	Vina-A01
Number of samples	10
AVG_Duration	7 mins 4s
AVG_RMSD (nm)	0.2024534
AVG_Correlation	0.475989

Table 3 Test Case #Vina-A01 (Averaged from 5 independent docking runs)

Test case #	Vina-A02
Number of samples	201
AVG_Duration	139 mins 42s
AVG_RMSD (nm)	0.3649292
AVG_Correlation	0.5114483406

Table 4 Test Case #Vina-A02 (Averaged from 5 independent docking runs)

6.2.1 Task 1

Objective: Apply PSO to AutoDock Vina and works effectively and maintain accuracy not worse than Vina.

By the preliminary tests, PSOxVina ran without any stopping criterion. For 201 complexes, it will take almost a day. In this task, just for ensuring the PSO running correctly, I decide to test only 10 complexes. Figure 21 shows those 10 complexes name with its torsion size.

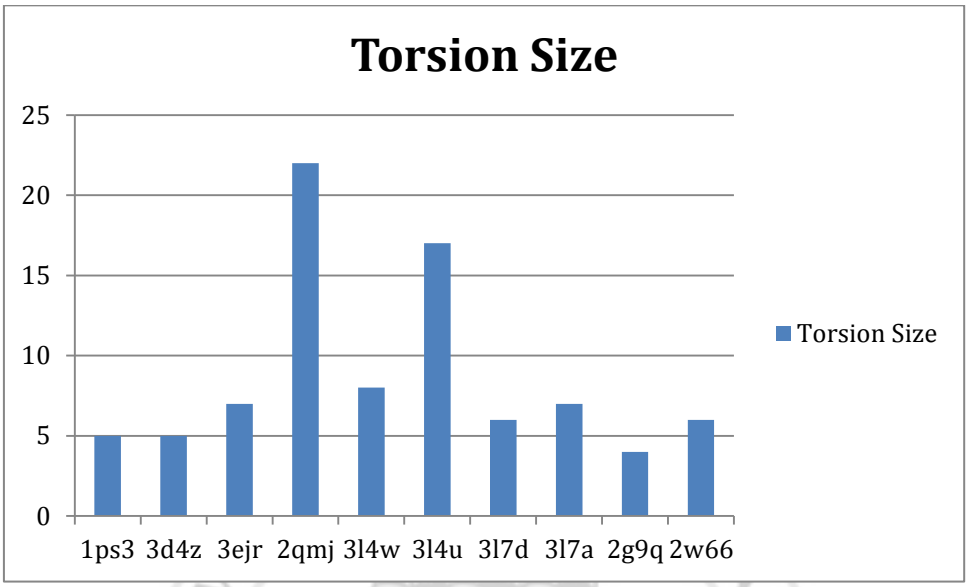


Figure 21: Torsion size distribution of 10 randomly selected complexes

Test case #	F101
Number of samples	10
C1	2.5
C2	2.5
W	1
# of birds	10
Duration	87 mins 32s
RMSD (nm)	0.139304
Correlation	0.48617390

Table 5 Test Case #F101
(Vina-A01: Duration:7 mins 4s , RMSD (nm): 0.2024534, Correlation: 0.475989)

Table 5 shows the docking result of PSOxVina with 10 complexes. Compare with VINA, my initial point has improved the accuracy compared with Vina. However, this stage hasn't speeded up the docking process yet.

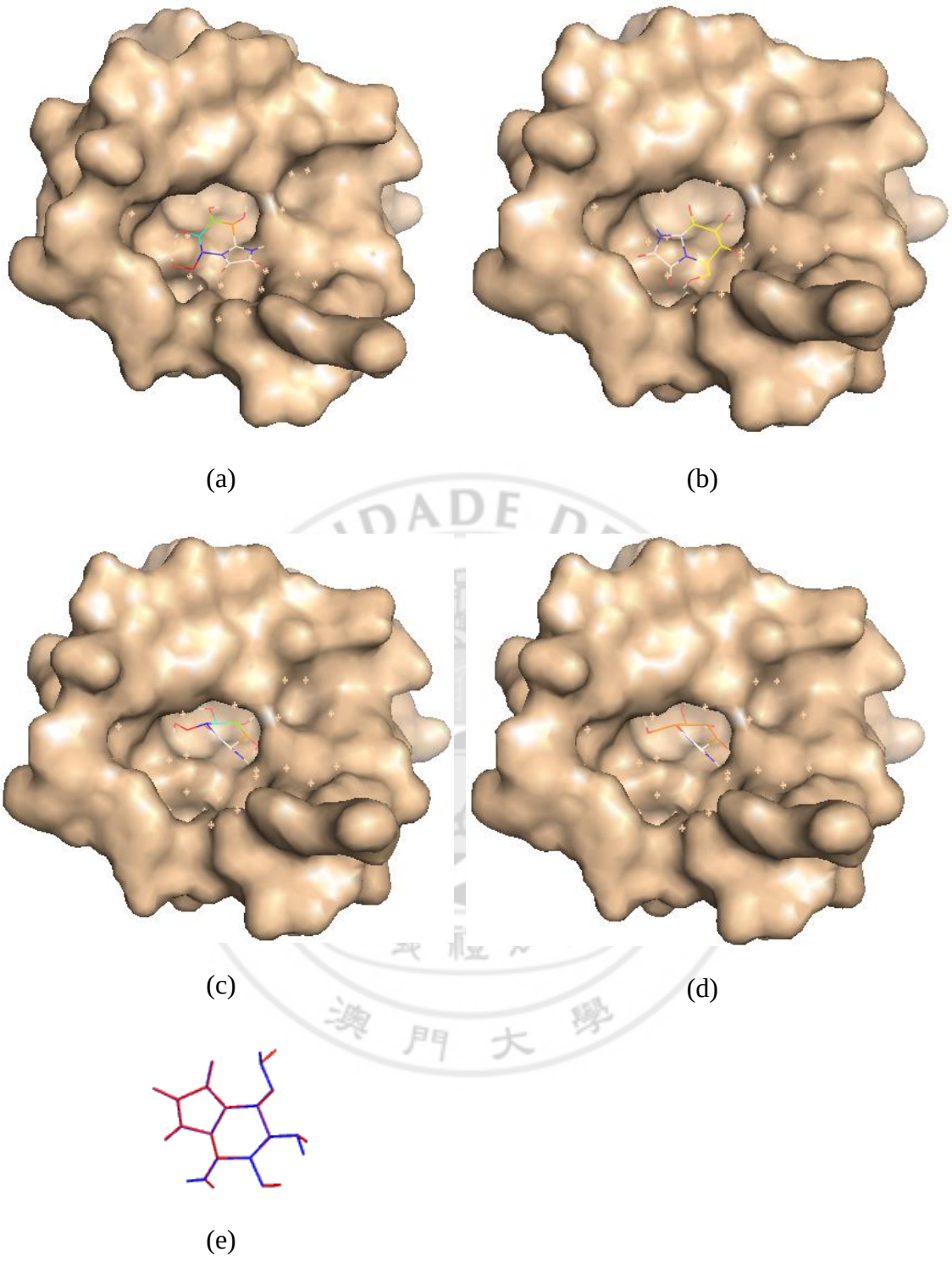
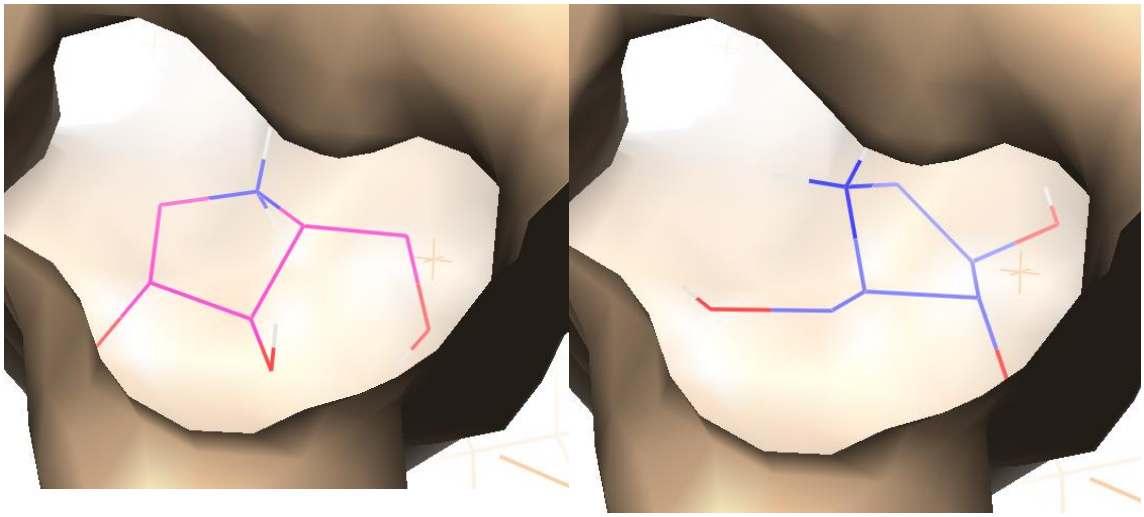
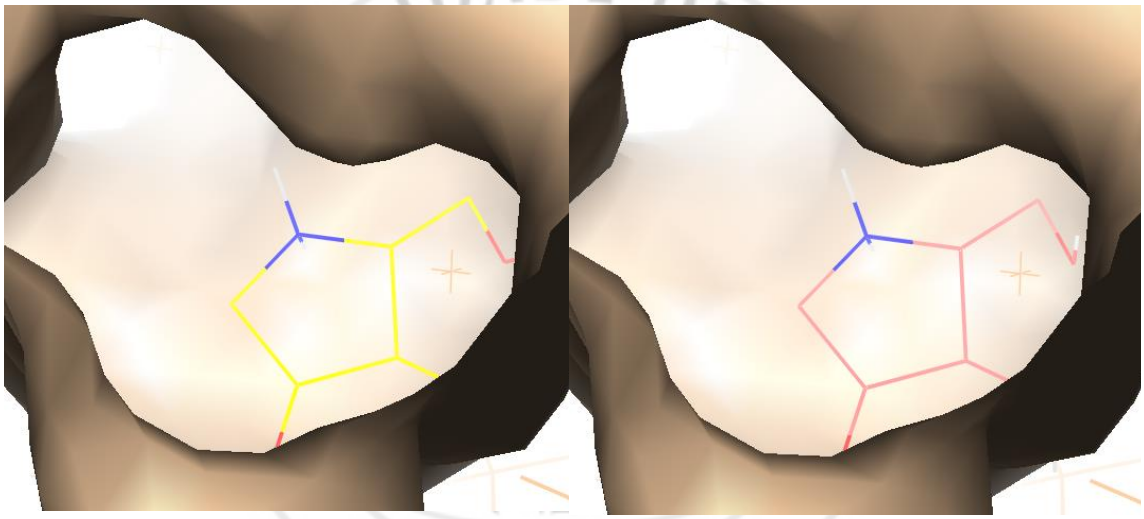


Figure 22 1ps3 Docking process snapshots (a)1st iteration (b)100th iteration (c)1000th iteration (d)8000th iteration (e) Overlap with original Vina result (highlight with red)

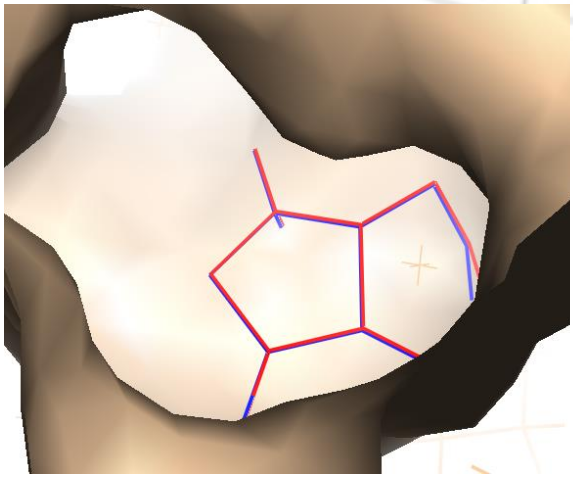
Figure 22 shows 1ps3 protein-ligand complex docking process snapshots that captured in different iterations, from beginning, middle and ending. Figure 22(e) overlap with original Vina docked conformation, obviously the result almost same as Vina.



(a) (b)



(c) (d)



(e)

Figure 23 2g9q Docking process snapshots (a)1st iteration (b)100th iteration (c)1000th iteration (d)8000th iteration (e) Overlap with original Vina result(highlight with red)

Figure 23 shows 2g9q protein-ligand complex docking process snapshots that captured in different iterations, from beginning, middle and ending. Figure 22 (e) overlap with original Vina docked conformation, obviously from Figure 22 and Figure 23 the PSO is applied successfully and run correctly. In the next task, I will focus on looking for the opportunities for improving much better both in boost up and accuracy.

6.2.2 Task 2

Objective: Tuning parameter, including w, c1, c2 and number of birds against the docking time and accuracy

Test case #	F201	F202	F203	F204	F205
Number of samples	10	10	10	10	10
C1	2.5	2.5	2.5	2.5	2.5
C2	2.5	2.5	2.5	2.5	2.5
W	1	1	1	1	1
# of birds	9	8	12	11	7
Duration	79mins 57s	72mins 12s	102mins 5s	94mins 18s	65mins 1s
RMSD (nm)	0.135824	0.118366	0.174542	0.162415	0.135545
Correlation	0.52033309	0.47869168	0.47800366	0.47908974	0.47566165

Table 6 Test Case #F201~#F205
(Vina-A01: Duration:7 mins 4s , RMSD (nm): 0.2024534, Correlation: 0.475989)
(#F101: Birds: 10, Duration: 87mins 32s, RMSD (nm): 0.139304, Correlation: 0.4861739)

Table 6 summarize 5 test cases based on different number of birds. From 7 birds to 12 birds in each thread. The data gives evidence that by the number of birds decrease, the spending time will also reduce. On the other hand, test case F202 and F205 RMSD are better than Vina with lower time spending, F202 looks very promising and move a little bit closer to this project objective. F202 will be selected to test with different w, c1 and c2 parameters. For the reason not choosing F205, it is because F205 boost up 9% of the system but lose 14% of accuracy. Eventually I pick F202 instead of F205 to balance the trade-off.

Refer to [3], the author find an alternative velocity equation and constraints(see Equation 9, Equation 10 and Equation 11), in that paper the result shows the birds will converge much more fast than before. In this parameters test, the equation will be implemented and #F202 parameters will be used to calculate the K. By the multiplication associative law, K actually is the weight of v_i and the weight also applies to c1 and c2. Hence, w, c1 and c2 will be pre-calculated then assign to the program.

$$v_i(t+1) = K*[v_i(t) + c1*rand() * (pbest_i(t) - x_i(t)) + c2 * rand() * (gbest(t) - x_i(t))] \quad \textit{Equation 9}$$

$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4}\right|} \quad \textit{Equation 10}$$

$$\varphi = c1 + c2 > 4 \quad \textit{Equation 11}$$

Test case #	F202	F206	F207	F208	F209
Number of samples	10	10	10	10	10
C1	2.5	0.66	0.33	0.99	0.26
C2	2.5	0.66	0.33	0.99	0.26
W	1	0.26	0.13	0.36	0.66
# of birds	8	8	8	8	8
Duration	72mins 12s	60mins	57min 36s	56mins 19s	53mins
RMSD (nm)	0.118366	0.15756	0.107608	0.122352	0.110873
Correlation	0.47869168	0.44025626	0.50249733	0.4689542	0.47296169

Table 7 Test Case #F202, #F206~#F209
(Vina-A01: Duration:7 mins 4s , RMSD (nm): 0.2024534, Correlation: 0.475989)

Table 7 shows different combination of parameters docking result. The lowest RMSD is test case #F207 RMSD has improved 9% compared with #F202. #F209 just improved 6.3%. #F206 and #F208 has decreased 33.1% and 3.37% respectively. These test cases also generate converge at which iteration (see Table 8). This is important to have such kind of data, the data gives an idea for discussing the stopping criterion in next task and further optimization. I defined if the best conformation keeps no change or the fitness value keeps changing < 0.0001 with 350 steps then

may consider all the particles are converge to the best solution. The number of steps was observed by preliminary test, most of the fitness value only changes very little or even no change after several steps then the ‘several steps’ would be that number.

	F202	F206	F207	F208	F209
1ps3	7851	491	672	617	781
3d4z	17896	652	666	507	685
3ejr	14088	682	12381	801	881
2qmj	3252	2769	36303	9679	8595
3l4w	4674	715	1066	1009	830
3l4u	28756	1350	8953	4015	1629
3l7d	14508	524	655	653	753
3l7a	2056	1118	617	680	831
2g9q	16372	525	745	645	677
2w66	10076	526	695	1597	1107
Average	11952.9	935.2	6275.3	2020.3	1676.9

Table 8 Test Case #F202, #F206~#F209 with converge iteration number

Compare to the best one which I chose from the first task. #F206, #F208 and #F209 converge 92.17%, 85% and 85.97% faster compare with #F202.

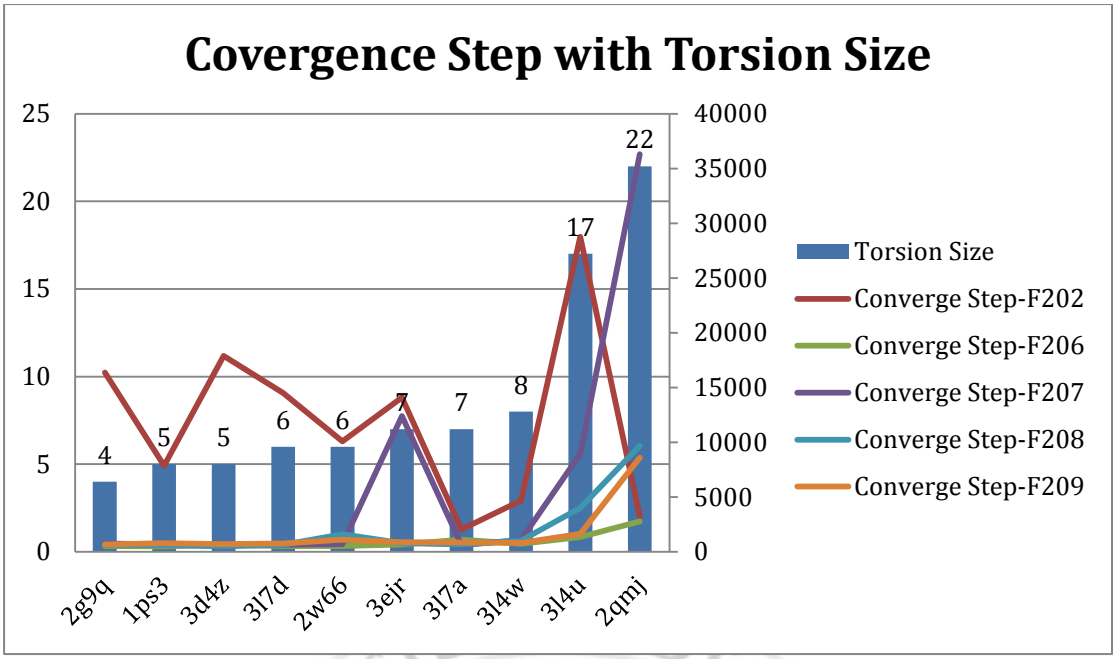


Figure 24 Cases from Table 7 compare converge step and torsion size (Complex name has been arranged in small-large order)

Figure 24 gives an intuitive illustration, F206, F208 and F209 converge almost at the same step in most of the complexes. Only start from 17 torsion size, the converge step begins to grow up. 10 test samples may not powerful enough to state the trend, I will pick F206, F208 and F209 to dock with 201 complexes which are the whole core set of PDBbind.

Test case #	F206	F208	F209
Number of samples	201	201	201
C1	0.66	0.99	0.26
C2	0.66	0.99	0.26
W	0.26	0.36	0.66
# of birds	8	8	8
Duration	1223mins 54s	1249mins 17s	1221mins
RMSD (nm)	0.307343	0.297607	0.30006
Correlation	0.51504922	0.52736977	0.52450163

Table 9 Test Case #F206-1, #F208-1 and #F209-1
(Vina-A02: Duration:139 mins 42s , RMSD (nm): 0.3649292, Correlation: 0.5114483406)

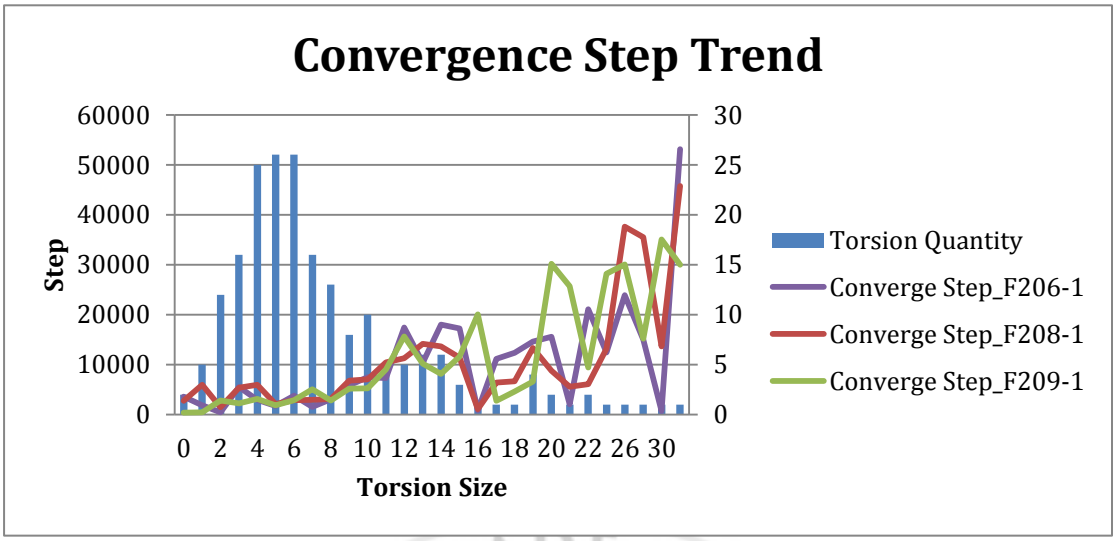


Figure 25 Convergence trend of test case in Table 9 (Last convergence appear)

Figure 25 shows the convergence trend of Table 9. The convergence steps were captured in the latest convergence step, and further take the average by the same number of torsion. From Figure 25, those trends look close to each other, except after 20 torsions and the convergence step increases by the number of torsion grows. The specific numbers of overall average convergence steps of different test cases is shown in Table 10, the best is #F209-1. However Figure 26 is the convergence trend of Table 9 but captured in the first convergence appeared. Figure 26 states the different result compare with Figure 25, the differences show me, even the first convergence appear, PSO still can find a better result. This time #F208-1 is the best which converge rapidly. To further apply the stopping criterion, case #F208-1 and #F209-1 will be selected.

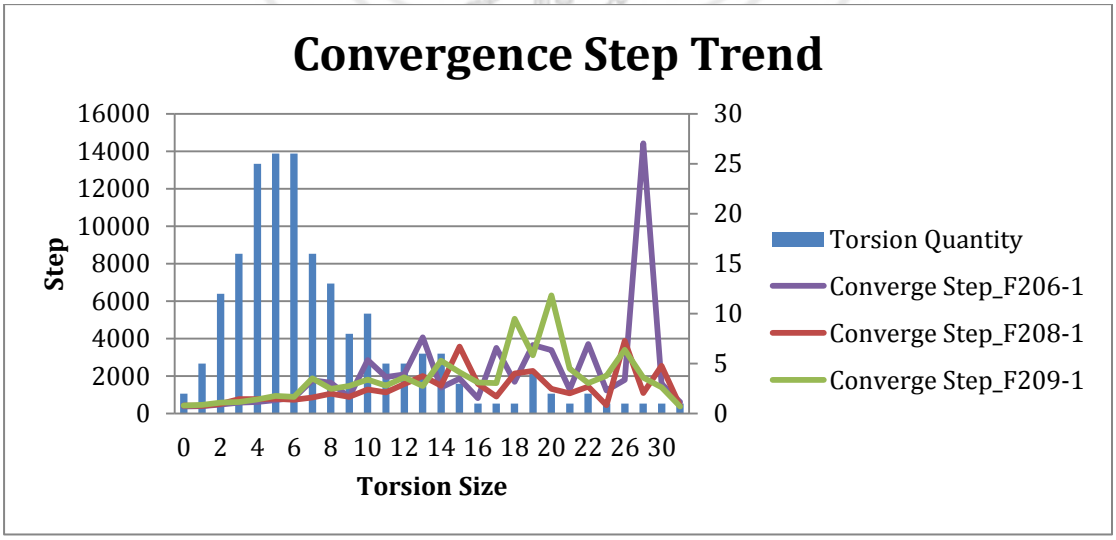


Figure 26 Convergence trend of test case in Table 9 (First convergence appear)

	F206-1	F208-1	F209-1
Last convergence	5600	5934	5249
First convergence	1363	1010	1303

Table 10 Specific number of overall average converge step

6.2.3 Task 3

Objective: Aim to find the most suitable stopping criterion and parameters by analysis the test cases.

To boost up the docking process, quickly convergence is required. #F208-1 and #F209-1 give the outstanding convergence step when convergence first appears. Hence, the goal of criterion try to stop at the first convergence appears. From the previous section, a condition of finding the first convergence and last convergence has been declared: if the best conformation keeps no change or the fitness value keeps changing < 0.0001 with 350 steps then may consider all the particles are converge to the best solution.

Test case #	F301	F302
Number of samples	201	201
C1	0.26	0.99
C2	0.26	0.99
W	0.66	0.36
# of birds	8	8
Duration	96mins 36s	85mins 56s
RMSD (nm)	0.3301812	0.3360118
Correlation	0.509694248	0.506732804

Table 11 Test Case #F301 and #F302 (Averaged from 5 independent docking runs)
(Vina-A02: Duration:139 mins 42s , RMSD (nm): 0.3649292, Correlation: 0.5114483406)

CHAPTER 7. Discussion

After series different parameter combinations of test cases in the previous Chapter, I settle down the candidate parameters on #F302. To consider the possible improvement, I try to look at how did the best test case perform in accuracy.

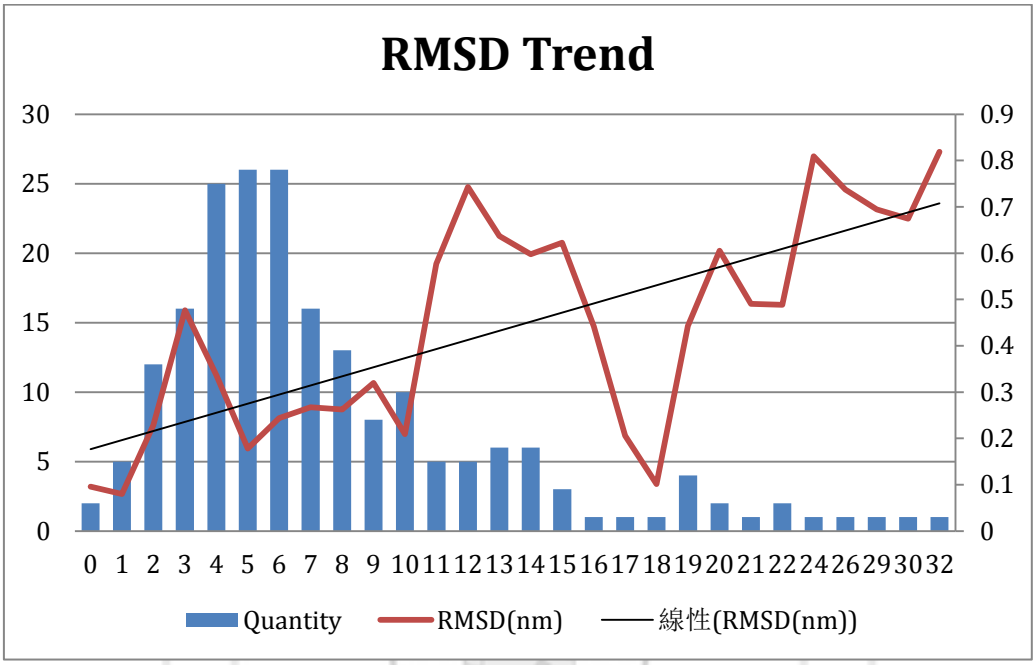


Figure 27 #F302 RMSD trend

Figure 27 shows the trend of the RMSD against different torsion size of ligand. Seems RMSD increasing by the torsion size grow, even there is a big drop in 18, I think it is caused by rarely samples. If the RMSD increasing can't avoided, or need to be traded by time, then I choose saving the time without keeping accurate. It is because in CADD, they need rapidly program with a roughly solution. The possible to reduce the time is decrease number of birds when they docked with large torsion size of ligand. So I base on 8 birds, to calculate number of birds for different torsion size of ligand by

$$N_i = \left\lfloor (8 - \frac{t_i}{10}) \right\rfloor$$ ***Equation 12***

where *i* is the ligand, *N* is the number of birds, *t* represent torsion size of ligand.

Test case #	F401	Vina-A02
Number of samples	201	201
C1	0.99	--
C2	0.99	--
W	0.36	--
# of birds	See Equation 12	--
Duration	75mins 56s	139mins 42s
RMSD (nm)	0.3391788	0.3649292
Correlation	0.504714078	0.5114483406

Table 12 Test Case #F401 and #Vina-A02 (Averaged from 5 independent docking runs)

Table 12 shows the docking result after implemented the Equation 12. The docking time performs as same as expected, decrease 11.7%. But RMSD makes me surprise that only increase 0.9%. It is worth to trade the accuracy for boost up the docking process. Compare with Autodock Vina, RMSD improves 7% and boost up 46% of the docking process. Figure 28 shows the ratio of the best independent docking run for which which RMSD < 0.2nm. #F401 achieves 55.72% a little bit win Vina 53.73% Finally, I settle down #F401 would be this project result and name as PSOxVina. The detail docking result please refer to CHAPTER 10.1 Independent Docking Result

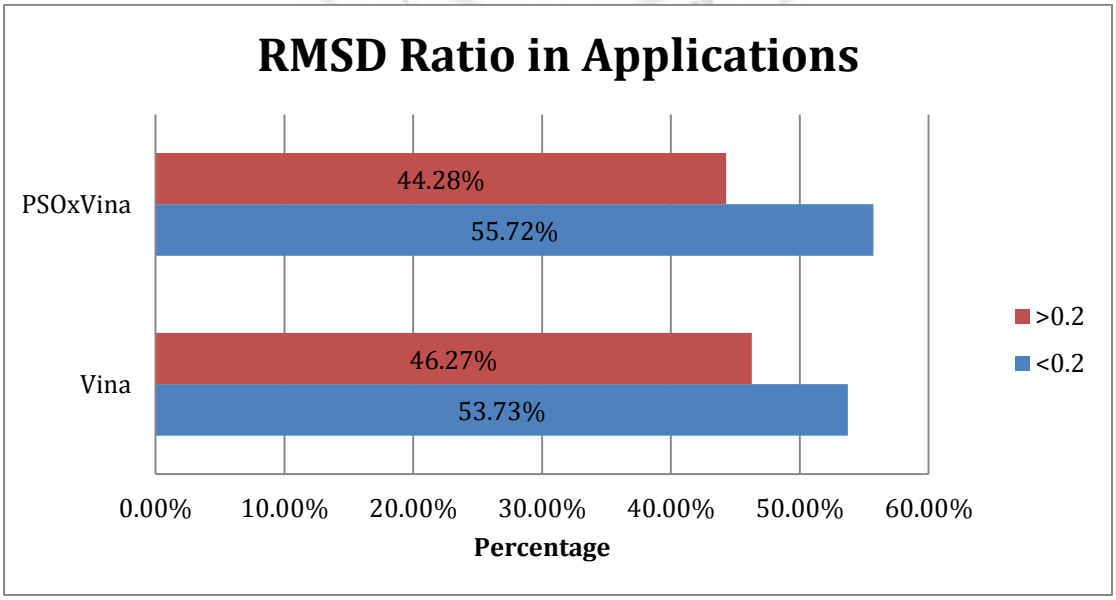


Figure 28 Ratio of RMSD < 0.2nm by PSOxVina and Vina

CHAPTER 8. Conclusion

PSOxVina, the first apply PSO to Vina docking program, has based on the advantage of Vina to further improve accuracy and to be able to search quickly and intelligently over the huge conformational space. An automated pipe-line also performs well while I also using the script for running those test cases. This project may consider success according to the goal I set in the beginning. It gives me an unexpected result. The accuracy also improves while I try to boost up the system.

8.1 Future Work

It is still far away to being perfect, even the docking time has been boosted up. As mentioned the most time consuming process is the local search. After discussed with my supervisor Dr. Shirely, we thought there would be several approaches that may also able to boost up the time. (1) Using GPU to computing the local search, it is not a diffcult solution but looks promising to boost up the process, CUDA (Compute Unified Device Architecture) [12] was introduced by NVIDIA Corporation. CUDA provides a set of library to developer for GPU programming. There is an exisiting research paper has successfully implemented BFGS to GPU computing [13]. By my review, the implemenatin boost up ~95% with very little (e^{-11}) energy changed. (2) Optimize the PSO parameters, from thoses test cases and analysis, different combination of parameters can affect the accuracy and converge time. (3) Find stopping criterion that situable for all size of ligand is essential. It is because by my research, different criteria make the docking time change as well as the accuracy.

CHAPTER 9. REFERENCES

[1] T. Food, “Fact Sheet New Drug Development Process.”

[2] C. Kumar, “An Insight to Drug Designing by In Silico approach in Biomedical Research,” *jphmr.com*, vol. 1, no. 2, pp. 63–65, 2013.

[3] V. Namasivayam and R. Günther, “PSO@ AUTODOCK: A fast flexible molecular docking program based on swarm intelligence,” *Chem. Biol. Drug Des.*, vol. 70, no. 6, pp. 475–484, Dec. 2007.

[4] Y. Liu, L. Zhao, W. Li, D. Zhao, M. Song, and Y. Yang, “FIPSDock: a new molecular docking technique driven by fully informed swarm optimization algorithm.,” *J. Comput. Chem.*, vol. 34, no. 1, pp. 67–75, Jan. 2013.

[5] G. ALGORITHM, “Crypto-Arithmetic Problem using Parallel Genetic Algorithm (PGA),” *Citeseer*, vol. 2, 2012.

[6] J. Kennedy and R. Eberhart, “Particle swarm optimization,” *... IEEE Int. Conf. neural ...*, vol. 4, pp. 1942–1948, 1995.

[7] S.-Y. Huang, S. Z. Grinter, and X. Zou, “Scoring functions and their evaluation methods for protein-ligand docking: recent advances and future directions.,” *Phys. Chem. Chem. Phys.*, vol. 12, no. 40, pp. 12899–908, Oct. 2010.

[8] S. Handoko, X. Ouyang, and C. Su, “QuickVina: Accelerating AutoDock Vina Using Gradient-Based Heuristics for Global Optimization,” *IEEE/ACM Trans. ...*, vol. 9, no. 5, pp. 1266–1272, 2012.

[9] T. Pdbbind and P. D. Bank, “A Brief Introduction to the PDBbind Database v . 2013,” pp. 1–2, 2013.

[10] R. Huey, G. M. Morris, and S. Forli, “Using AutoDock 4 and AutoDock Vina with AutoDockTools : A Tutorial,” 2012.

[11] wwPDB, “wwPDB Format version 3 Coordinate Section.” [Online]. Available: http://www.wwpdb.org/documentation/format33/sect9.html.

[12] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with CUDA,” *portal.acm.org*, 2007.

[13] Y. Fei, G. Rong, B. Wang, and W. Wang, “Parallel L-BFGS-B Algorithm on GPU,” *Comput. Graph.*, no. 2, 2014.

CHAPTER 10. Appendix

10.1 Independent Docking Result

10.1.1 AutoDock Vina

	Vina-A02-1	Vina-A02-2	Vina-A02-3	Vina-A02-4	Vina-A02-5
1ps3	0.047739	0.047263	0.046494	0.048249	0.048578
3d4z	0.101343	0.101978	0.101158	0.100205	0.094814
3ejr	0.102806	0.260712	0.06456	0.106114	0.268602
2qmj	0.21083	0.210669	0.211202	0.210703	0.211117
3l4w	0.191515	0.07375	1.40502	0.191335	1.39316
3l4u	0.186663	0.827089	0.144804	0.810046	1.57177
3l7d	0.136647	0.136719	0.137281	0.135955	0.135964
3l7a	0.068067	0.170225	0.066674	0.067778	0.066922
2g9q	0.052689	0.047937	0.0489	0.048988	0.048469
2w66	0.036203	0.501951	0.038389	0.035619	0.03733
2wca	0.370158	0.379586	0.362392	0.381201	0.381493
2vvn	0.029483	0.024911	0.026458	0.025625	0.028815
3sjf	0.088051	0.088285	0.08519	0.063205	0.0662
2xej	0.532767	0.498572	0.531305	0.551181	0.460903
2xeg	0.953167	0.936717	0.935583	0.363843	0.926322
2x96	1.20595	1.20312	1.20399	1.15713	1.20186
2x91	0.725322	0.720346	0.71176	0.774202	0.711498

1j37	0.129996	0.068714	0.128739	0.130676	0.1278
3cj2	0.041453	0.042748	0.04097	0.042718	0.040505
2d3u	0.037389	0.038125	0.036165	0.03646	0.040537
3gnw	0.030987	0.030692	0.030264	0.03193	0.030027
1gpk	0.023774	0.026129	0.024046	0.02928	0.024679
1h23	0.977842	0.145594	0.983175	0.645826	0.993722
1e66	0.032031	0.034539	0.031695	0.034706	0.033378
3f3a	0.190117	0.181476	0.191141	0.193294	0.180068
3f3c	0.096078	0.09607	0.093145	0.09615	0.093922
3f3e	0.169075	0.17061	0.169987	0.169011	0.168722
2rkm	0.095622	0.098346	0.078618	0.106551	0.101013
1b9j	0.127172	0.129028	0.123468	0.128072	0.127248
1b7h	0.096124	0.098406	0.110506	0.111873	0.101549
2j77	0.037541	0.038247	0.033808	0.034013	0.036725
2j78	0.034759	0.037265	0.034898	0.039585	0.039249
2cet	0.822751	0.248763	0.823557	0.821609	0.822353
2zxd	0.045869	0.040794	0.042615	0.043084	0.040225
2zwz	0.048857	0.047835	0.053047	0.054298	0.050727
2zx6	0.838459	0.067886	0.063012	0.078449	0.068454
3bra	0.936674	0.936536	1.33688	0.942056	0.942966
3ckp	0.913649	0.056518	0.065942	0.913288	0.065492

2g94	0.416093	0.267485	0.271433	0.51387	0.971864
3fk1	0.752731	0.726078	0.737042	0.725211	0.733963
2qft	0.068808	0.063518	0.744326	0.744497	0.741676
2pq9	0.131895	0.139074	0.120524	0.132264	0.139382
2qwb	0.480071	0.091293	0.091179	0.480096	0.090453
2qwd	0.095113	0.096887	0.103191	0.10879	0.075115
2qwe	0.036683	0.047232	0.059568	0.077279	0.384443
1n2v	0.262926	0.26556	0.263677	0.263516	0.262885
1r5y	0.019986	0.029166	0.032912	0.032478	0.025223
3ge7	0.088823	0.088597	0.090123	0.089402	0.088458
3hec	0.051446	0.051047	0.051585	1.01266	0.049634
2zb0	0.133623	0.134083	0.134358	0.134783	0.134371
3e93	0.039948	0.040055	1.37526	0.038126	1.47603
3gv9	1.25646	1.26011	1.2563	1.25646	1.25651
2pu2	0.742819	0.744469	0.744041	0.743338	0.745247
1xgj	0.378737	0.380717	0.380306	0.379274	0.380184
1q8t	0.780874	0.781627	0.779479	0.781136	0.781815
1xh6	0.753347	0.087618	0.087761	0.087725	0.087634
1re8	0.122729	0.034266	0.034324	0.033655	0.034094
3h30	0.527923	0.527542	0.538882	0.529522	0.528294
2zjw	0.223962	0.223754	0.22251	0.2221	0.223708

3pe2	0.0509	0.059731	0.05606	0.052467	0.098183
2v00	0.13419	0.134622	0.134974	0.139582	0.138592
5er2	0.949567	0.933933	0.445567	0.950558	0.447446
4er2	1.00509	0.373412	0.449619	1.32592	0.408022
2wec	1.13286	1.05185	1.08807	1.11949	1.11809
1bxq	0.268433	0.253829	0.249816	0.605634	0.264012
1bxo	0.049174	0.04876	0.048442	0.048685	0.049961
2brb	0.108001	0.109483	0.107967	0.113907	0.107416
3jvs	0.302724	0.301869	0.302667	0.302619	0.302414
1nvq	0.521963	0.521952	0.521898	0.521886	0.522011
3mfv	0.166245	0.176872	0.173138	0.174459	0.184302
3f80	0.167155	0.166398	0.148342	0.150943	0.129152
3kv2	0.093498	0.119337	0.098036	0.099185	0.119528
1nwl	0.799223	0.666329	0.880881	0.80781	0.853709
1c88	1.11284	1.10988	1.10898	0.093371	1.11599
2qbp	0.111002	0.111468	0.111616	0.111401	0.111402
3fcq	0.922889	0.922864	0.922052	0.921856	0.921494
1os0	0.546355	0.542096	0.547459	0.548353	0.539921
4tmn	1.34102	1.29561	1.28877	0.752698	1.34788
2fzc	0.517572	0.525537	0.526348	0.522471	0.521264
2h3e	0.089079	0.088709	0.088668	0.089196	0.088735

1d09	0.090734	0.142769	0.142782	0.142866	0.142921
3kgq	0.251936	0.474105	0.473858	0.460561	0.163808
2rfh	0.168856	0.121615	0.166719	0.120837	0.166966
6cpa	0.832428	0.702904	0.753871	0.839726	0.867797
2exm	0.57304	0.566671	0.566332	0.603757	0.569828
1b39	0.280005	0.316735	0.317289	0.330309	0.317414
2xmy	0.869248	0.878904	0.859117	0.879351	0.852802
1qi0	1.09527	1.09572	1.09608	1.09492	1.09341
1w3k	0.029661	0.033057	0.034841	0.034338	0.03481
1w3l	0.080393	0.080321	0.081111	0.080699	0.080468
1bcu	0.069054	0.069194	0.069815	0.069664	0.06995
1c1v	0.847243	0.847214	0.845449	0.840027	0.846906
1sl3	0.068116	0.067341	0.073391	0.07023	0.068917
3imc	0.463436	0.463425	0.463557	0.463247	0.46336
3iub	0.436246	0.438554	0.440522	0.4437	0.442589
3cow	0.060788	0.061391	0.061257	0.060855	0.060329
3b3s	0.159124	0.158803	0.161006	0.160281	0.159065
1ft7	0.201888	0.201932	0.22856	0.242224	0.22963
1txr	0.275738	0.648428	0.650714	0.262938	0.18655
3acw	0.170975	0.170786	0.134114	0.170069	0.171116
2zcr	0.341643	0.180517	0.338942	0.334663	0.333557

2zcq	0.609631	0.558772	0.556053	0.613299	0.558075
1y1z	0.119672	0.117941	0.116478	0.119666	0.119345
1pb8	1.17317	0.826363	1.17349	1.17282	0.214356
1pbq	0.082023	0.08189	0.082585	0.083077	0.082618
1lvu	0.108754	0.104843	0.102878	0.065048	0.066058
1v48	0.098983	0.099819	0.096653	0.115374	0.137611
1b8o	0.02743	0.027425	0.027738	0.025677	0.028783
3adv	1.65649	1.65568	1.65589	1.65233	1.65542
2i4j	1.14889	0.957846	0.810165	0.954538	1.14407
2p4y	0.081809	0.997364	0.09264	0.772382	0.775219
3mhw	0.481183	0.482478	0.48133	0.481237	0.481026
1o5b	0.097944	0.098238	0.097991	0.098583	0.098294
1sqa	0.148146	0.254498	0.590242	0.224559	0.173399
3kme	0.30758	0.309119	0.30838	0.047043	0.274432
3b92	0.119109	0.358969	0.501596	0.199599	0.151521
3e8r	0.29304	0.293918	0.292831	0.291264	0.292929
2osf	0.433984	0.433946	0.433965	0.43427	0.434143
2pow	0.534696	0.534773	0.534794	0.534623	0.534558
1if7	0.424288	0.406501	0.250485	0.348417	0.415757
2xdl	0.031103	0.032525	0.033598	0.027573	0.036581
1yc1	0.080327	0.080438	0.08111	0.080592	0.080431

2yki	0.023621	0.024294	0.023338	1.014	0.024933
1p1q	0.525836	0.525969	0.526355	0.525265	0.526139
3bfu	0.27945	0.496705	1.64118	1.51727	1.64276
1ftm	0.23561	0.041913	0.483793	0.48399	0.042161
2yhd	2.02811	2.02863	2.02777	1.96064	2.02782
1z95	0.102156	0.102915	0.101406	0.10248	0.102302
3g0w	0.022374	0.022275	0.021931	0.020261	0.021083
1vso	0.175342	0.184804	0.187193	0.174742	0.202753
3gbb	0.08616	1.30979	0.079134	0.080061	0.084043
3fv1	0.075308	1.38743	0.074908	0.487159	1.41525
2b1v	0.087348	0.087163	0.086013	0.087763	0.086327
2qe4	0.062028	0.061913	0.061856	0.060543	0.375756
2p15	1.56935	0.748682	0.748762	0.032331	0.034143
2y5h	0.212967	0.196932	0.225056	0.220145	0.205158
2xbv	0.05356	0.056742	0.074115	0.075248	0.074616
1mq6	0.17459	0.17586	0.126963	0.126095	0.175497
1loq	0.105755	0.108785	0.106305	0.105403	0.106389
1lol	0.371749	0.373696	0.375047	0.374594	0.374045
1x1z	0.102104	0.101644	0.10259	0.102343	0.102478
4tim	0.162467	0.18049	0.162978	0.133839	0.183266
2v2h	0.169576	0.187712	0.168901	0.169574	0.14564

1trd	0.136558	0.428442	0.139208	0.140159	0.140914
1uto	1.43126	1.43143	1.43669	1.43223	1.43139
3gy4	0.1342	0.133442	0.134541	0.134971	0.135059
1o3f	0.16945	0.168701	0.169171	0.168117	0.161697
1jys	0.168794	0.287071	0.287139	0.287073	0.168133
1nc1	0.119364	0.119577	0.119568	0.118497	1.27048
1y6q	1.65583	1.67828	1.67508	0.043444	0.04907
3pce	0.50463	0.505056	0.505142	0.505161	0.504387
3pcn	0.503583	0.502568	0.503279	0.502432	0.503768
3pcj	0.146657	0.144444	0.145765	0.110562	0.146956
2pgz	0.033161	0.037193	0.035862	0.450747	0.033466
2wn9	0.732232	0.732674	0.73296	0.732064	0.726006
2x00	0.051595	0.050929	0.050893	0.05025	0.050521
2r23	0.448358	0.45197	0.465311	0.468335	0.39768
2bmk	0.139314	0.142251	0.09927	0.138743	0.135709
1kel	0.169502	0.148051	0.141789	0.169623	0.114857
3ozt	0.109441	0.109721	0.043465	0.11061	0.109879
3oe5	0.09027	0.098289	0.112084	0.076478	0.105245
3nw9	0.030925	0.124462	0.055137	0.040095	0.056204
1zea	0.923698	0.881058	1.06029	1.00157	1.02961
2cgr	0.030279	0.037418	0.031144	0.030195	0.028516

1igj	0.199094	0.194891	0.193262	0.191552	0.191904
1lbk	0.908757	0.918781	1.07275	0.720543	0.718666
2gss	0.327191	0.327768	0.328865	0.327345	0.328134
3ie3	0.300421	0.334415	0.301179	0.309135	0.310024
3n7a	0.031873	0.034612	0.033078	0.031612	0.861034
3n86	1.64375	0.100136	0.100574	0.099346	0.099847
2y71	0.024451	0.025084	0.026159	0.026559	0.024199
3lka	0.575579	0.576782	0.559516	0.558597	0.561315
3ehy	0.080146	0.081632	0.076564	0.427787	0.129975
3f17	0.431776	0.432053	0.432395	0.432436	0.431623
1adl	0.491264	0.582617	0.649178	0.336707	0.452103
1g74	0.615201	0.611679	0.506384	0.183317	0.618315
2nnq	0.084167	0.087724	0.085794	0.085392	0.085684
1str	0.250327	0.274773	0.748419	0.289891	0.881389
1swr	1.19403	0.096374	0.06704	0.065968	0.098062
3rdo	0.099989	0.058683	0.093576	0.096794	0.097748
3jvk	1.0156	1.01136	1.0192	1.02624	0.998907
3u5j	0.020945	0.025081	0.025033	0.022189	0.020949
3mxf	0.069725	0.069562	0.069597	0.069581	0.069336
3cft	0.556377	0.556547	0.555857	0.556248	0.556303
3nex	0.275501	0.290976	0.326781	0.322105	0.320351

2g5u	0.258174	0.257527	0.257901	0.258587	0.257525
3dxg	0.610246	0.610364	0.611801	0.610934	0.610504
1o0m	0.115687	0.098742	0.115214	0.116187	0.115383
1u1b	0.711994	0.725575	0.697744	0.70802	0.722805
1sv3	0.32391	0.323988	0.323569	0.271378	0.240015
1jq8	0.994112	0.976192	0.988819	1.00171	1.05554
2arm	0.476355	0.471315	0.470819	0.492353	0.471736
3ov1	0.101946	0.111529	0.116116	0.079272	0.122221
3s8o	0.541113	0.548866	0.54843	0.546306	0.550759
1jyq	0.323712	0.334826	0.330652	0.964493	0.309705
1a30	0.735536	0.740898	0.910641	0.735884	0.713182
3cyx	0.085215	0.084645	0.084667	0.084574	0.084711
2i4x	0.185141	0.244891	0.76911	0.426774	0.236634
1d7j	0.185526	0.184575	0.184547	0.18599	0.184201
1fki	0.023048	0.023056	0.023048	0.023052	0.023056

Table 13 5 independent runs of Vina-A02

10.1.2 PSOxVina

	F401-1	F401-2	F401-3	F401-4	F401-5
1ps3	0.047441	0.045039	0.04754	0.048706	0.045562
3d4z	0.099087	0.100899	0.100547	0.054506	0.101795
3ejr	0.045015	0.258736	0.283398	0.260119	0.259335
2qmj	0.209237	0.211613	0.204862	0.212455	0.213891

3l4w	0.190621	0.191913	0.190724	0.190783	1.4098
3l4u	0.166297	0.145635	0.204088	0.154987	0.161979
3l7d	0.120907	0.161264	0.135596	0.120804	0.143621
3l7a	0.066695	0.066954	0.067073	0.066812	0.065493
2g9q	0.046794	0.052054	0.050615	0.048507	0.04956
2w66	0.033731	0.037227	0.036366	0.032867	0.03958
2wca	0.381014	0.380688	0.37124	0.351658	0.721269
2vvn	0.026693	0.028745	0.026283	0.026313	0.027284
3sjf	0.077984	0.054847	0.071432	0.087132	0.070839
2xej	0.610423	0.529154	0.574133	0.546816	0.568119
2xeg	0.613241	0.338183	0.657913	0.645991	0.593018
2x96	1.07928	1.19575	0.830526	1.20239	1.19385
2x91	0.738823	0.770069	0.785771	0.711435	1.09083
1j37	0.608804	0.101307	0.531175	0.609221	0.612498
3cj2	0.042944	0.65143	0.041417	0.651212	0.039793
2d3u	0.038722	0.983247	0.10518	0.960003	0.092852
3gnw	0.030665	0.029832	0.030598	0.02987	0.029294
1gpk	0.022981	0.023861	0.026323	0.02533	0.028483
1h23	0.313033	0.986385	1.17343	0.266651	0.105721
1e66	0.032985	0.033258	0.031697	0.033562	0.033955
3f3a	0.177909	0.116188	0.176524	0.179357	0.175795

3f3c	0.03223	0.093978	0.024616	0.024309	0.024036
3f3e	0.16893	0.168991	0.133827	0.170528	0.13158
2rkm	0.07519	0.089796	0.809338	0.226251	0.099246
1b9j	0.121752	0.115359	0.108738	0.139203	0.1107
1b7h	0.099308	0.104684	0.104723	0.096465	0.149758
2j77	0.243477	0.033738	0.246334	0.244789	0.24439
2j78	0.036827	0.039919	0.03747	0.038439	0.038348
2cet	0.824883	0.819739	0.224724	0.23932	0.824811
2zxd	0.041955	0.042307	0.041518	0.040884	0.042
2zwz	0.052184	0.048639	0.049108	0.047647	0.051757
2zx6	0.065914	0.062974	0.065072	1.60629	0.064884
3bra	0.947344	1.33618	1.33642	1.33745	1.33883
3ckp	0.206818	0.055295	0.106477	0.266403	0.921421
2g94	0.133773	0.265642	0.797538	0.769859	0.780196
3fk1	0.724917	0.726216	0.72744	0.728179	0.75801
2qft	0.737596	0.509199	0.589687	0.590948	0.726533
2pq9	0.089707	0.100901	0.119329	0.120046	0.025246
2qwb	0.365332	0.48962	0.054642	0.056556	0.134576
2qwd	0.364238	0.068566	0.098204	0.078269	0.066436
2qwe	0.054802	0.072914	0.112058	0.112448	0.054023
1n2v	0.264414	0.267126	0.376957	0.26491	0.376857

1r5y	0.030326	0.019906	0.020474	0.02782	0.025287
3ge7	0.089071	0.087332	0.089222	0.089547	0.088931
3hec	0.057732	0.053884	0.049561	0.053562	0.053942
2zb0	0.076268	0.07648	0.075884	0.134289	0.07697
3e93	0.038837	0.039261	0.038411	0.039009	0.039682
3gv9	1.2556	1.25964	1.25664	1.25959	1.25668
2pu2	0.743012	0.744775	0.745737	0.743798	0.742465
1xgj	0.226644	1.14296	0.949759	0.555431	1.21325
1q8t	0.77916	0.779223	0.779108	0.779146	0.779061
1xh6	0.055323	0.055229	0.078689	0.078965	0.087598
1re8	0.034768	0.085306	0.143885	0.033714	0.033331
3h30	0.37874	0.378924	0.379097	0.200008	0.378268
2zjw	0.223171	0.630083	0.630313	0.223351	0.630223
3pe2	0.074211	0.792118	0.067892	0.046495	0.065651
2v00	0.084434	0.083627	0.089943	0.083817	0.090212
5er2	1.00015	0.979431	0.231489	0.885203	0.465175
4er2	1.01318	0.37487	1.00578	0.912169	1.32061
2wec	0.521785	0.452759	0.538354	1.1048	1.13757
1bxq	0.261142	0.264797	0.598825	0.231147	0.267388
1bxo	0.036238	0.04901	0.049564	0.072695	0.072291
2brb	0.108607	0.106635	0.113051	0.107819	0.588269

3jvs	0.296732	0.303126	0.302887	0.302965	1.02849
1nvq	0.522058	0.522107	0.522057	0.522098	0.521858
3mfv	0.169414	0.193669	0.467112	0.222989	0.218159
3f80	0.188896	0.136556	0.131132	0.106811	0.140612
3kv2	0.131216	0.101844	0.100746	0.101158	0.101987
1nwl	0.883376	0.920316	0.788259	0.763869	0.904928
1c88	1.13746	1.11302	1.11538	0.051487	1.1106
2qbp	0.099	0.099093	0.099161	0.11148	0.111515
3fcq	0.163215	0.143249	0.921397	0.143071	0.163568
1os0	0.57349	0.56404	0.55975	0.105735	0.861755
4tmn	1.28661	0.904034	0.724144	0.744852	1.22565
2fzc	0.53157	0.519222	0.528839	0.531051	0.522304
2h3e	0.035562	0.088355	0.035597	0.089154	0.099406
1d09	0.042001	0.118611	0.04243	0.143043	0.042479
3kgq	0.473799	0.47391	0.468497	0.473819	0.46583
2rfh	0.09546	0.165986	0.165259	0.1413	0.147762
6cpa	0.632374	0.748763	0.755785	0.613919	0.982145
2exm	0.565693	0.65439	0.565745	0.579951	0.653018
1b39	0.328096	0.329313	0.329596	0.328299	0.330363
2xmy	0.381387	0.458397	0.843	0.826293	0.917901
1qi0	0.837684	1.12558	0.820738	0.799195	1.09633

1w3k	0.840048	0.839942	0.834278	0.536272	0.068972
1w3l	0.065855	1.02559	0.584027	0.047165	0.611788
1bcu	0.589627	0.068866	0.069408	0.069141	0.068707
1c1v	0.839472	0.845313	0.845732	0.845673	0.845198
1sl3	0.468727	0.850027	0.067614	0.972469	0.067521
3imc	0.464912	0.463339	0.46335	0.023835	0.021491
3iub	0.4421	0.167794	0.441689	0.442832	0.167604
3cow	0.345461	0.061464	0.349099	0.061255	0.351755
3b3s	0.157687	0.156727	0.15742	0.160516	0.156824
1ft7	0.242531	0.263017	0.242775	0.194626	0.185957
1txr	0.155915	0.746511	0.249224	0.203081	0.223445
3acw	0.133015	0.170225	0.132541	0.130613	0.132824
2zcr	0.141189	0.109166	0.147712	0.130573	0.139406
2zcq	0.579255	0.546264	0.539889	0.622749	0.549807
1y1z	0.118053	0.012726	0.116767	0.028219	0.030363
1pb8	0.125776	0.08087	0.080529	0.123636	0.083145
1pbq	0.082211	0.081995	0.082442	0.035454	0.034959
1lvu	0.119047	0.114929	0.120071	0.109849	0.126227
1v48	0.103184	0.137883	0.152332	0.092479	0.084879
1b8o	0.026502	0.02666	0.0276	0.02808	0.026637
3adv	1.65458	1.65504	0.92776	1.58884	1.65486

2i4j	1.09516	0.8756	1.17449	0.515728	1.16103
2p4y	0.078372	0.105235	0.067031	0.050766	0.052266
3mhw	0.48477	0.166288	0.485181	0.483507	0.166674
1o5b	0.097476	0.098402	0.098377	0.097478	0.098124
1sqa	0.254273	0.252518	1.35953	0.14273	0.15975
3kme	0.045408	0.045948	0.309634	0.28674	0.212587
3b92	0.184264	0.152426	0.166191	0.198402	0.305336
3e8r	0.302562	0.29137	0.291087	0.291182	0.291288
2osf	0.433065	0.434079	0.433999	0.433958	0.433926
2pow	0.534705	0.534717	0.534543	0.534717	0.534702
1if7	0.446763	0.261404	0.39757	0.384792	0.250607
2xdl	0.032801	0.09004	0.031768	0.033905	0.086793
1yc1	0.076739	0.080611	0.080057	0.080234	0.080038
2yki	0.024143	0.024143	0.024143	0.024143	0.024143
1p1q	0.26146	0.517601	0.260903	0.521652	0.522322
3bfu	0.043291	0.040243	0.039603	0.038415	0.039786
1ftm	0.031747	0.488169	0.032733	0.482503	0.040794
2yhd	2.02821	2.02836	2.02789	2.02793	2.0279
1z95	0.062485	0.07168	0.061721	0.102071	0.100115
3g0w	0.019878	0.021703	0.022008	0.020347	0.020496
1vso	0.495952	0.18371	0.734504	0.564495	0.523262

3gbb	0.079396	0.043323	0.079835	0.07907	0.081164
3fv1	0.074191	0.075466	0.027841	0.026175	0.074273
2b1v	0.085695	0.086177	0.086663	0.085981	0.08641
2qe4	0.061592	0.060201	0.062487	0.060416	0.061695
2p15	0.032595	0.033149	0.033114	0.031935	0.032867
2y5h	0.045204	0.213226	0.205132	0.217057	0.119734
2xbv	0.118507	0.273504	0.074522	0.074405	0.055644
1mq6	0.143113	0.038364	0.131503	0.074272	0.17794
1loq	0.105525	0.106205	0.208665	0.673875	0.673723
1lol	0.373478	0.372205	0.377801	0.376784	0.386408
1x1z	0.105799	0.044291	0.101948	0.103213	0.102762
4tim	0.133181	0.181979	0.181419	0.106235	0.161146
2v2h	0.187791	0.169115	0.162374	0.164893	0.187128
1trd	0.181769	0.181867	0.186901	0.182973	0.177577
1uto	0.171927	1.29562	0.13392	0.116194	0.102421
3gy4	0.135272	0.133673	0.134805	0.133344	0.057727
1o3f	0.160773	0.150195	0.169111	0.161324	0.172313
1jys	0.168694	0.168268	0.168088	0.168576	0.167923
1nc1	0.119429	0.118629	0.035179	0.11904	0.115718
1y6q	0.045744	0.042219	0.0487	0.04664	0.046889
3pce	0.505288	0.503824	0.504381	0.505143	0.504473

3pcn	0.504359	0.503069	0.50708	0.506112	0.504146
3pcj	0.110679	0.14502	0.14404	0.111462	0.144848
2pgz	0.035372	0.034176	0.035302	0.035546	0.035288
2wn9	0.688756	0.667447	0.733329	0.683508	0.682895
2x00	0.051149	0.051491	0.051188	0.050164	0.050383
2r23	0.665934	0.378811	0.450919	0.604481	0.096094
2bmk	0.742764	0.564715	0.162739	0.129927	0.150087
1kel	0.190619	0.142795	0.104597	0.201122	0.162858
3ozt	0.047397	0.047235	0.04901	0.044243	0.755259
3oe5	0.076626	0.094928	0.872022	1.10195	0.053792
3nw9	0.120613	0.030985	0.0542	0.030006	0.899464
1zea	1.14607	0.927625	0.845539	1.00503	0.904954
2cgr	0.028539	0.063947	0.029385	0.028611	0.029107
1igj	0.194001	0.747749	0.19151	0.194491	0.262663
1lbk	0.86347	0.716926	0.692176	0.692709	0.81164
2gss	0.308858	0.32521	0.329281	0.327971	0.331653
3ie3	0.30149	0.316564	0.308627	0.307428	0.333198
3n7a	0.026968	0.030307	0.023302	0.029574	0.025831
3n86	0.073131	0.100625	0.141036	0.10057	0.099894
2y71	0.023965	0.023754	0.025468	0.024652	0.023034
3lka	0.558311	0.576199	0.558523	0.558633	0.558266

3ehy	0.364523	0.428754	0.10786	0.835711	0.359236
3f17	0.29984	0.264317	0.300133	0.299964	0.232066
1adl	0.311487	0.661338	0.267522	0.312907	0.366354
1g74	0.204818	0.603642	0.263032	0.385918	0.425068
2nnq	0.101886	0.064746	0.065463	0.06331	0.085559
1str	0.295238	0.554592	0.98242	0.718696	0.546531
1swr	0.065922	0.059053	0.063627	1.94007	1.27661
3rdo	0.061723	0.098083	0.098049	0.060724	0.097906
3jvk	1.06774	0.717507	0.890145	0.648827	0.963844
3u5j	0.021448	0.025223	0.020979	0.020988	0.020977
3mxf	0.041607	0.079702	0.069507	0.041142	0.041123
3cft	0.555902	0.556521	0.556207	0.556068	0.106687
3nex	0.317273	0.330798	0.317712	0.320812	0.316126
2g5u	0.739975	0.698795	0.042444	0.363853	0.702474
3dxg	0.581314	0.552961	0.582854	0.584175	0.582601
1o0m	0.114047	0.045578	0.098817	0.504581	0.114381
1u1b	0.321233	0.196468	1.05324	0.773847	0.244185
1sv3	0.225409	0.270829	0.240327	0.22231	0.242356
1jq8	0.912282	0.999078	0.947686	1.04487	1.157
2arm	0.534324	0.491723	0.472362	0.490895	0.508076
3ov1	0.152477	0.254392	0.134056	0.566745	0.094435

3s8o	0.173322	0.919864	0.878997	0.82331	0.624086
1jyq	1.04719	1.01405	0.920998	1.40512	0.479733
1a30	0.738402	0.716488	0.66036	0.735641	0.541558
3cyx	0.051027	0.084547	0.054958	0.054636	0.082251
2i4x	0.351729	0.232694	1.14533	0.324256	0.336977
1d7j	0.18678	0.341063	0.190212	0.142697	0.146194
1fki	0.023042	0.023065	0.023052	0.023065	0.023052

Table 14 5 independent runs of PSOxVina

10.2 Automated Pipe-line Scripts

10.2.1 run_vina.sh

```
#!/bin/bash

# utako: from Kami but modified options in prepare_ligand4 & prepare_receptor4


#TOOL_DIR=/Library/MGLTools/latest/MGLToolsPckgs/AutoDockTools/Utilities24

TOOL_DIR=/usr/local/MGLTools-1.5.7rc1/MGLToolsPckgs/AutoDockTools/Utilities24

VINA=/home/kami/autodock_vina_1_1_2/build/linux/release/vina


infile=2012_core_name.lst

count=0


for nameList in `grep -v "\#" $infile | cut -f1 -d\ ` ; do


        #nameList=3ov1

        #if [ $count -ge 1 ]; then

        # exit 0

        #fi

count=`expr $count + 1`
```

```
  echo "================================================"

  echo "================================================"

  echo "[$count]Processing data $nameList "

  echo "================================================"

  echo "================================================"




  cd $nameList

#if [ $count -ge 192 ]; then


rm ./rmsd2expt.log ./*.pdbqt ./*fit.pdb ./*randomize.pdb ./*vinadocked.pdb ./vina.log ./vina_score_only.log ./config.txt ./*_lig
and.pdb ./fit.log



  ${TOOL_DIR}/prepare_ligand4.py -l ./${nameList}_ligand.mol2 -o ./${nameList}_ligand.pdbqt -A 'hydrogens' -U
'nphs_lps_waters'

  ${TOOL_DIR}/prepare_receptor4.py -r ./${nameList}_protein.pdb -o ./${nameList}_protein.pdbqt -A 'hydrogens' -U
'nphs_lps_waters'




  cut -c-66 ${nameList}_ligand.pdbqt > ${nameList}_ligand.pdb  # pdb can be understood by GROMACS


  awk -f ../pdbbox.awk ${nameList}_pocket.pdb  > config.txt


  $VINA --receptor ${nameList}_protein.pdbqt --ligand ${nameList}_ligand.pdbqt --config config.txt --num_modes 1 --log
vina.log --cpu 8 --exhaustiveness 8



  # compare to expt structure

  cut -c-66 ${nameList}_ligand_out.pdbqt > ${nameList}_ligand_vinadocked.pdb

  echo -e "0\n0" |g_confrms -f1 ${nameList}_ligand.pdb -f2 ${nameList}_ligand_vinadocked.pdb -nofit > fit.log

  rmsd=`cat fit.log |grep lsq|awk '{ print $9 }'`

  echo "RMSD to expt structure ${nameList}: $rmsd"

  echo $rmsd > rmsd2expt.log

#fi

  cd ../
```

```
  #read ans

done


exit 0
```

10.2.2 collect_score.sh

```
#!/bin/bash



infile=2012_core_data.lst

count=0


grep -v "#" $infile | awk '{ print  $4 " "}' > t1.tmp


if [ -e t2.tmp ]; then

   rm t2.tmp

fi


touch t2.tmp


for nameList in `grep -v "\#" $infile | cut -f1 -d\ `; do


  count=`expr $count + 1`

  echo "Processing data ${count}. $nameList "


  cd $nameList

  score=` tail vina.log|grep 0.000|awk '{ print $2 }' | head -n 1`

 # score=`tail -2 vina.log|grep 1|awk '{ print $2 }'`

  echo "--$score"

  cd ../


  echo "$score" >> t2.tmp

done
```

```
paste -d\  t1.tmp t2.tmp > vina_score.dat

rm t1.tmp t2.tmp

exit 0
```

10.2.3 p.pro

```
function readtable,filename

  cmd="wc "+filename

  spawn,cmd,res

  cmd="head -1 "+filename +"| awk '{ print NF }'"

  spawn,cmd,nrcol

  data=dblarr(nrcol[0],res[0])

  openr,1,filename & readf,1,data & close,1

  return,data

end


pro p


old_X=!X

old_Y=!Y

old_P=!P

!X.STYLE=1       ;; 1: exact axis to the data range

!Y.STYLE=1

;!X.TICKINTERVAL=20 ;; interval/range b/w major tickmarks

;!Y.TICKINTERVAL=5

!X.MINOR=2       ;; # of minor intervals, 0: default, -1: suppress

!Y.MINOR=2

!X.TICKLEN=0.02

!Y.TICKLEN=0.02

!X.THICK=7       ;; axis thickness

!Y.THICK=7       ;;

!P.THICK=7.0     ;; thickess of the line connecting points, better 3.0

!P.CHARSIZE=1.5   ;; font size, e.g. 1.8
```

```
!P.CHARTHICK=5.0   ;; font thickness

title='!3'

xtitle='!3time (ns)'   ;; !3 standard character set

angstrom = '!3' + STRING(197B) + '!X'

ytitle='!3area per lipid (nm!E2!N)'


set_plot,'ps'

loadct,13


device,filename="p.eps",/ENCAPSULATED,/color

data=readtable("vina_score.dat")

vinascore=data[1,*]

RT=8.3144621*300.0/1000.0

ki=exp(vinascore*4.184/RT)

pki=-1.0*alog10(ki)



fitted=linfit(data[0,*],pki)

a=fitted[0]

b=fitted[1]

print,fitted

pearson=correlate(data[0,*],pki)

print,pearson

plot,data[0,*],pki,psym=SYM(1),xtitle="experimental binding affinity",ytitle="predicted binding
affinity",/isotropic,xrange=[1,12],yrange=[1,12]

oplot,[0,13],[a,a+b*13.0],thick=3

xyouts,1.5,11,"correlation ="+strcompress(string(pearson))


device,/close


!X=old_X

!Y=old_Y

!P=old_P
```

```
stop

end
```

10.3 Source Code of PSOxVina

10.3.1 mutate.cpp

```
/*

   Copyright (c) 2006-2010, The Scripps Research Institute

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.

   Author: Dr. Oleg Trott <ot14@columbia.edu>,
         The Olson Lab,
         The Scripps Research Institute

*/

#include "mutate.h"
#include <iostream>

sz count_mutable_entities(const conf& c) {
        sz counter = 0;
```

```
	VINA_FOR_IN(i, c.ligands)
		counter += 2 + c.ligands[i].torsions.size();
	VINA_FOR_IN(i, c.flex)
		counter += c.flex[i].torsions.size();
	return counter;
}

// does not set model
void mutate_conf(conf& c, const model& m, fl amplitude, rng& generator) { // ONE OF: 2A for position, similar amp for orientation, randomize torsion
	std::cout<< "Enter the normal mutate" << '\n';
	sz mutable_entities_num = count_mutable_entities(c);
	if(mutable_entities_num == 0) return;
	int which_int = random_int(0, int(mutable_entities_num - 1), generator);
	VINA_CHECK(which_int >= 0);
	sz which = sz(which_int);
	VINA_CHECK(which < mutable_entities_num);

	VINA_FOR_IN(i, c.ligands) {
		if(which == 0) { c.ligands[i].rigid.position += amplitude * random_inside_sphere(generator); return; }
		--which;
		if(which == 0) {
			fl gr = m.gyration_radius(i);
			if(gr > epsilon_fl) { // FIXME? just doing nothing for 0-radius molecules. do some other mutation?
				vec rotation;
				rotation = amplitude / gr * random_inside_sphere(generator);
				quaternion_increment(c.ligands[i].rigid.orientation, rotation);
			}
			return;
		}
		--which;
		if(which < c.ligands[i].torsions.size()) { c.ligands[i].torsions[which] = random_fl(-pi, pi, generator); return; }
```

```
                which -= c.ligands[i].torsions.size();
        }
        VINA_FOR_IN(i, c.flex) {
                if(which < c.flex[i].torsions.size()) { c.flex[i].torsions[which] = random_fl(-pi, pi, generator); return; }
                which -= c.flex[i].torsions.size();
        }
}

// does not set model

void mutate_conf(output_type& candidate, const model& m, fl amplitude, rng& generator, pso* particle,const precalculate& p ,const igrid& ig,change& g,const vec& v,quasi_newton& quasi_newton_par,int step) { // ONE OF: 2A for position, similar amp for orientation, randomize torsion
        //conf c = candidate.c;
        sz mutable_entities_num = count_mutable_entities(candidate.c);
        if(mutable_entities_num == 0) return;
        int which_int = random_int(0, int(mutable_entities_num - 1), generator);
        VINA_CHECK(which_int >= 0);
        sz which = sz(which_int);
        VINA_CHECK(which < mutable_entities_num);


        int y;
        VINA_FOR_IN(i, candidate.c.ligands) {

                model tmp_m = m;
                const vec authentic_v(1000, 1000, 1000);
                //loop for each particle

                        //Take part the position (either position or orientation or torsion)
                        if(which == 0) {


                        for (y=0;y<particle->number;y++){
```

```
//evaulate the fitness function (calculate the energy)
//double energy = tmp_m.eval(p,ig,v,c)-tmp_m.eval_intramolecular(p, authentic_v, c);   //remark on 2/4/2014

candidate.c.ligands[i].rigid.position = particle->getCurrentPosition(y);
candidate.c.ligands[i].rigid.orientation = particle->getCurrentOrientation(y);
for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
        candidate.c.ligands[i].torsions[z] = particle->getCurrentTorsion(i,z);

//double energy = tmp_m.eval_deriv(p, ig, v, candidate.c, g); //(const precalculate& p, const igrid& ig, const vec& v, const conf& c, change& g)
//double energy = tmp_m.eval(p,ig,v,candidate.c)-tmp_m.eval_intramolecular(p, authentic_v, candidate.c);
//if(energy < 0)
quasi_newton_par(tmp_m, p, ig, candidate, g, v);
//else
//candidate.e = energy;
//std::cout<<"Energy calculated by quasi newton:"<< candidate.e << "Energy Calculate by formula:" << energy << '\n';

//set the personal best(energy value and position);
if(candidate.e < particle->getPersonalBest(y))
{
        //quasi_newton_par(tmp_m, p, ig, candidate, g, v);
        particle->updatePersonalBest(y,candidate.e);

        particle->updateBestPosition(y,candidate.c.ligands[i].rigid.position);
        particle->updateBestOrientation(y,candidate.c.ligands[i].rigid.orientation);
        for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
                particle->updateBestTorsion(y, candidate.c.ligands[i].torsions[z],z);
        //set the global best(energy value and position);
        if(candidate.e < pso::gbest_fit)
```

```
                                        {
                                                std::cout << "current_P:" << candidate.e << "
quasi_e:"<<candidate.e << "     the current best_P:" << pso::gbest_fit<<"    Current number steps" <<step <<'\n';
                                                particle->updateGlobalBest(y);

                                                pso::gbest_position = candidate.c.ligands[i].rigid.position;
                                                pso::gbest_orientation =
candidate.c.ligands[i].rigid.orientation;
                                                for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
                                                        pso::gbest_torsion[z] =
candidate.c.ligands[i].torsions[z];
                                        }
                                }

                                //update each particle in every dimension
                                particle->updateVelocity(generator,y);
                                //compute the new position;
                                particle->computeNewPositions(y);
                        }

                        for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
                                candidate.c.ligands[i].torsions[z] = pso::gbest_torsion[z];
                        candidate.c.ligands[i].rigid.orientation = pso::gbest_orientation;
                        candidate.c.ligands[i].rigid.position = pso::gbest_position;
                                //candidate.e = pso::gbest_fit;
                                return;
                        }



                        --which;
                        //Take part orientation (either position or orientation or torsion)
                        if(which == 0) {
                                fl gr = m.gyration_radius(i);
```

```
if(gr > epsilon_fl) { // FIXME? just doing nothing for 0-radius molecules. do some other mutation?

for (y=0;y<particle->number;y++){

//evaulate the fitness function (calculate the energy)

//double energy1 = tmp_m.eval(p,ig,v,c)-tmp_m.eval_intramolecular(p, authentic_v, c);

//I saw this formula in main.cpp

candidate.c.ligands[i].rigid.position = particle->getCurrentPosition(y);

candidate.c.ligands[i].rigid.orientation = particle->getCurrentOrientation(y);

for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)

candidate.c.ligands[i].torsions[z] = particle->getCurrentTorsion(i,z);

//double energy = tmp_m.eval_deriv(p, ig, v, candidate.c, g); //(const precalculate& p, const igrid& ig, const vec& v, const conf& c, change& g)

//double energy = tmp_m.eval(p,ig,v,candidate.c)-tmp_m.eval_intramolecular(p, authentic_v, candidate.c);

//if(energy < 0)

quasi_newton_par(tmp_m, p, ig, candidate, g, v);

//else

//candidate.e = energy;

//std::cout<<"Energy calculated by quasi newton:"<< candidate.e << "Energy Calculate by formula:" << energy << '\n';

//double energy1 = candidate.e;

//set the personal best(energy value and position);

if(candidate.e < particle->getPersonalBest(y))

{

//quasi_newton_par(tmp_m, p, ig, candidate, g, v);

particle->updatePersonalBest(y,candidate.e);
```

```
                                                    particle-
>updateBestPosition(y,candidate.c.ligands[i].rigid.position);

                                                    particle-
>updateBestOrientation(y,candidate.c.ligands[i].rigid.orientation);

                                                    for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)

                                                            particle->updateBestTorsion(y,
candidate.c.ligands[i].torsions[z],z);



                                                    //set the global best(energy value and position);

                                                    if(candidate.e< pso::gbest_fit)//pso::gbest_fitForO)

                                                    {


                                                            std::cout << "current_O:" << candidate.e << "
        quasi_e:"<<candidate.e <<"      the current best_O:" << pso::gbest_fit<<"    Current number steps"<<step<<'\n';


                                                    particle->updateGlobalBest(y);



                                                    pso::gbest_position                             =
candidate.c.ligands[i].rigid.position;

                                                    pso::gbest_orientation                          =
candidate.c.ligands[i].rigid.orientation;

                                                    for(int
z=0;z<candidate.c.ligands[i].torsions.size();z++)

                                                            pso::gbest_torsion[z]                   =
candidate.c.ligands[i].torsions[z];

                                                    }
                                            }



                                            //update each particle in every dimension

                                            particle->updateVelocityO(generator,y);

                                            //compute the new position;

                                            particle->computeNewOrientation(y);

                                            }

                                    }
```

```
                                        for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
                                                candidate.c.ligands[i].torsions[z] = pso::gbest_torsion[z];
                                        candidate.c.ligands[i].rigid.orientation = pso::gbest_orientation;
                                        candidate.c.ligands[i].rigid.position = pso::gbest_position;
                                        return;
                                }

                                /*Torsions*/
                                --which;
                                if(which < candidate.c.ligands[i].torsions.size()) {

                                        for (y=0;y<particle->number;y++){


                                        //evaulate the fitness function (calculate the energy)
//                                      double energy = tmp_m.eval_deriv(*p, *ig, *v, c, *g); //(const precalculate& p, const igrid& ig, const vec& v, const conf& c, change& g)
                                        //double energy2 = tmp_m.eval(p,ig,v,c)-tmp_m.eval_intramolecular(p, authentic_v, c);
                                                candidate.c.ligands[i].rigid.position = particle->getCurrentPosition(y);
                                                candidate.c.ligands[i].rigid.orientation = particle->getCurrentOrientation(y);
                                                for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
                                                        candidate.c.ligands[i].torsions[z] = particle->getCurrentTorsion(i,z);
                                                //double energy = tmp_m.eval_deriv(p, ig, v, candidate.c, g); //(const precalculate& p, const igrid& ig, const vec& v, const conf& c, change& g)
                                                //double energy = tmp_m.eval(p,ig,v,candidate.c)-tmp_m.eval_intramolecular(p, authentic_v, candidate.c);
                                        //      if(energy < 0)
                                                quasi_newton_par(tmp_m, p, ig, candidate, g, v);
                                        //      else
                                        //      candidate.e = energy;
                                                //std::cout<<"Energy calculated by quasi newton:"<< candidate.e << "Energy Calculate by formula:" << energy << '\n';
                                                //double energy2 = candidate.e;
```

```
                                        //set the personal best(energy value and position);
                                        if(candidate.e < particle->getPersonalBest(y))
                                        {
                                                //quasi_newton_par(tmp_m, p, ig, candidate, g, v);
                                                particle->updatePersonalBest(y,candidate.e);
                                                for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
                                                        particle->updateBestTorsion(y,
candidate.c.ligands[i].torsions[z],z);                                  particle-
>updateBestPosition(y,candidate.c.ligands[i].rigid.position);
                                                particle-
>updateBestOrientation(y,candidate.c.ligands[i].rigid.orientation);

                                                //set the global best(energy value and position);
                                                if(candidate.e < pso::gbest_fit)//pso::gbest_fitForT[which])
                                                {
                                                        std::cout << "current_T:" << candidate.e << "
        quasi_e:"<<candidate.e<< "      the current best_T:" << pso::gbest_fit<<"     Current number step"<<step <<'\n';
                                                        particle->updateGlobalBest(y);

                                                        pso::gbest_position = candidate.c.ligands[i].rigid.position;
                                                        pso::gbest_orientation                          =
candidate.c.ligands[i].rigid.orientation;
                                                        for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
                                                                pso::gbest_torsion[z]                   =
candidate.c.ligands[i].torsions[z];
                                                }
                                        }

                                        //update each particle in every dimension
                                        particle->updateVelocityT(generator,y,which);
                                        //compute the new position;
                                        particle->computeNewTorsion(y,generator,which);
                                        }
```

```
                                        for(int z=0;z<candidate.c.ligands[i].torsions.size();z++)
                                                candidate.c.ligands[i].torsions[z] = pso::gbest_torsion[z];
                                        candidate.c.ligands[i].rigid.orientation = pso::gbest_orientation;
                                        candidate.c.ligands[i].rigid.position = pso::gbest_position;

                                        return;
                                }
                                which -= candidate.c.ligands[i].torsions.size();
                        }

        VINA_FOR_IN(i, candidate.c.flex) {
                        if(which < candidate.c.flex[i].torsions.size()) { candidate.c.flex[i].torsions[which] = random_fl(-pi, pi,
generator); return; }
                        which -= candidate.c.flex[i].torsions.size();
        }
}
```

10.3.2 monte_carlo.cpp

```
/*
HETATM   27  H   STI   202     10.733  94.264  65.567  1.00  0.00

  Copyright (c) 2006-2010, The Scripps Research Institute

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
  See the License for the specific language governing permissions and

  limitations under the License.


  Author: Dr. Oleg Trott <ot14@columbia.edu>,

        The Olson Lab,

        The Scripps Research Institute


*/


#include "monte_carlo.h"

#include "coords.h"

#include "quasi_newton.h"

#include "mutate.h"


output_type monte_carlo::operator()(model& m, const precalculate& p, const igrid& ig, const precalculate& p_widened, const igrid& ig_widened, const vec& corner1, const vec& corner2, incrementable* increment_me, rng& generator) const {

        output_container tmp;

        this->operator()(m, tmp, p, ig, p_widened, ig_widened, corner1, corner2, increment_me, generator); // call the version that produces the whole container

        VINA_CHECK(!tmp.empty());

        return tmp.front();

}


bool metropolis_accept(fl old_f, fl new_f, fl temperature, rng& generator) {

        if(new_f < old_f) return true;

        const fl acceptance_probability = std::exp((old_f - new_f) / temperature);

        return random_fl(0, 1, generator) < acceptance_probability;

}


void monte_carlo::single_run(model& m, output_type& out, const precalculate& p, const igrid& ig, rng& generator) const {

        conf_size s = m.get_size();

        change g(s);

        vec authentic_v(1000, 1000, 1000);
```

```
	out.e = max_fl;
	output_type current(out);
	quasi_newton quasi_newton_par; quasi_newton_par.max_steps = ssd_par.evals;

	//comment by kami
	//std::cout<< "enter single run" << '\n';

	VINA_U_FOR(step, num_steps) {
		output_type candidate(current.c, max_fl);

		//Enter key point
		mutate_conf(candidate.c, m, mutation_amplitude, generator);

		quasi_newton_par(m, p, ig, candidate, g, hunt_cap);
		if(step == 0 || metropolis_accept(current.e, candidate.e, temperature, generator)) {
			quasi_newton_par(m, p, ig, candidate, g, authentic_v);
			current = candidate;
			if(current.e < out.e)
				out = current;
		}
	}
	quasi_newton_par(m, p, ig, out, g, authentic_v);
}

void monte_carlo::many_runs(model& m, output_container& out, const precalculate& p, const igrid& ig, const vec& corner1,
const vec& corner2, sz num_runs, rng& generator) const {
	conf_size s = m.get_size();
	VINA_FOR(run, num_runs) {
		output_type tmp(s, 0);
		tmp.c.randomize(corner1, corner2, generator);
		single_run(m, tmp, p, ig, generator);
		out.push_back(new output_type(tmp));
	}
```

```
	out.sort();
}

output_type monte_carlo::many_runs(model& m, const precalculate& p, const igrid& ig, const vec& corner1, const vec& corner2, sz num_runs, rng& generator) const {
	output_container tmp;
	many_runs(m, tmp, p, ig, corner1, corner2, num_runs, generator);
	VINA_CHECK(!tmp.empty());
	return tmp.front();
}

// out is sorted
void monte_carlo::operator()(model& m, output_container& out, const precalculate& p, const igrid& ig, const precalculate& p_widened, const igrid& ig_widened, const vec& corner1, const vec& corner2, incrementable* increment_me, rng& generator) const {
	vec authentic_v(1000, 1000, 1000); // FIXME? this is here to avoid max_fl/max_fl
	conf_size s = m.get_size();
	change g(s);
	output_type tmp(s, 0);
	tmp.c.randomize(corner1, corner2, generator);  //first randomize
	fl best_e = max_fl;
	quasi_newton quasi_newton_par; quasi_newton_par.max_steps = ssd_par.evals;
	int birds_factor = (int)(tmp.c.ligands[0].torsions.size()/10);
	//init pso (argument:number of particle,range of space,seed)
	pso particle(birds_factor > 1 ? (int)(8/birds_factor) : 8 ,corner1,corner2,generator,tmp.c);
	//pso particle(8,corner1,corner2,generator,tmp.c);
	//std::cout<< max_fl << '\n';
	double energy=0;
	int count=0;
	//main loop
	//int tmp_step = 0;
	std::cout<< "Number of steps::::::" << num_steps << "    Torsion_size"<< tmp.c.ligands[0].torsions.size() <<'\n';
	VINA_U_FOR(step, num_steps) {
```

```
	//std::cout<< "Current of steps:::::" << step << '\n';
			if(increment_me)
				++(*increment_me);
			output_type candidate = tmp;

			//particle.print();
			//orginial::mutate_conf(candidate.c, m, mutation_amplitude, generator);
			mutate_conf(candidate, m, mutation_amplitude, generator,&particle,p,ig,g,hunt_cap,quasi_newton_par,step); //for each particle loop
			//std::cout<<"Finish mutate" <<'\n';
				//candidate.c.ligands[0].rigid.position = pso::gbest_position;
				//candidate.c.ligands[0].rigid.orientation = pso::gbest_orientation;

			//quasi_newton_par(m, p, ig, candidate, g, hunt_cap);

			if(step == 0 || metropolis_accept(tmp.e, candidate.e, temperature, generator)) {
				tmp = candidate;
				m.set(tmp.c); // FIXME? useless?

				// FIXME only for very promising ones
				if(tmp.e < best_e || out.size() < num_saved_mins) {
					quasi_newton_par(m, p, ig, tmp, g, authentic_v);
					m.set(tmp.c); // FIXME? useless?
					tmp.coords = m.get_heavy_atom_movable_coords();
					add_to_output_container(out, tmp, min_rmsd, num_saved_mins); // 20 - max size
					if(tmp.e < best_e)
						best_e = tmp.e;
				}
			}

/*
			std::string name = "./movie/test_out_";
```

```
                char num[20];
                sprintf(num,"%d",step);
                name += num;
                name += ".pdbqt";

                ofile f(path(name,boost::filesystem::native));
                m.write_model(f,1,"");
*/


                /***Criteria defined by Kami***/


                //if(std::abs(pso::gbest_fit - energy) < 0.0001*(num_steps / 10000))
                if(std::abs(pso::gbest_fit - energy) < 0.0001)
                {
                        //std::cout << "count" << count << '\n';
                        count +=1;
                        //double factor = 0.05/(num_steps / 10000);
                        //if(count >  500 - 200*std::pow(3/4,(int)(candidate.c.ligands[0].torsions.size()/10))  )
                        if(count > 350)
                        {
                                //tmp_step=(tmp_step == 0?  step : tmp_step);
                                //#std::cout << "Energy no changed since " << tmp_step << '\n';
                                std::cout << "Terminiated at " << step << '\n';
                                step = num_steps; //break the loop
                                count = 0;
                        }
                }else{
                        //tmp_step = 0;
                        energy = pso::gbest_fit;
                        count =0;
```

```
                }

        }

        std:: cout << pso::gbest_fit << '\n';

        VINA_CHECK(!out.empty());
        VINA_CHECK(out.front().e <= out.back().e); // make sure the sorting worked in the correct order
}
```

10.3.3 pso.cpp

```
#include "pso.h"
#include "random.h"

  // the vector of degree of freedom
        qt pso::gbest_orientation;
        fl* pso::gbest_torsion;
        vec pso::gbest_position;

        double pso::gbest_fit;

        pso::pso(int number,const vec corner1,const vec corner2, rng& g,conf& c){

                sz torsionSize = c.ligands[0].torsions.size();
                this->w = 0.36;
                this->R1Max_ = 1;
                this->R1Min_ = 0;
                this->R2Max_ = 1;
                this->R2Min_ = 0;
                this->c1 = 0.99;
                this->c2 = 0.99;
                double psi = c1+c2;
```

```
                //double k = 2 / (std::abs(2-psi-(std::sqrt(psi*psi-4))));
                double k =1;
                this->c1 *= k;
                this->c2 *= k;
                this->w *= k;
                this->g = g;
                this->corner1[0] = corner1[0];                  //minmum
                this->corner1[1] = corner1[1];
                this->corner1[2] = corner1[2];
                this->corner2[0] = corner2[0];                  //maximum
                this->corner2[1] = corner2[1];
                this->corner2[2] = corner2[2];
                this->torsionSize = (int)torsionSize;
                this->number = number;

                pso::gbest_torsion = new fl[torsionSize];

            init(g,c);

}

void pso::init(rng &g,conf& c)
{

            int i;
            for(i=0;i<this->number;i++)
            {
                        bird abc;
                        //set position part
                        abc.velocity = random_in_box(this->corner1,this->corner2,g);
                        abc.current_position = random_in_box(this->corner1,this->corner2,g);
                        //std::cout <<abc.current_position[0] << ':'<<abc.current_position[1]<< ':'
<<abc.current_position[2] <<'\n';
```

```
                abc.pbest_pos[0]=0;
                abc.pbest_pos[1]=0;
                abc.pbest_pos[2]=0;
                abc.pbest_fit = 1.7976931348623158e+308;
                abc.tmp_fit = 1.7976931348623158e+308;
                //set orientation part
                abc.vO = random_inside_sphere(g);
                qt tmp_o = c.ligands[0].rigid.orientation;
                quaternion_increment(tmp_o,  random_inside_sphere(g));
                abc.current_orientation = tmp_o;

                //init the array for the number of torsion
                abc.current_torsion=new fl[this->torsionSize];
                abc.vT=new fl[this->torsionSize];
                abc.pbest_torsion=new fl[this->torsionSize];

                for(int x=0;x<this->torsionSize;x++)
                {
                        abc.vT[x] = random_fl(-pi, pi, g);
                        abc.current_torsion[x] = random_fl(-pi, pi, g);
                }

                particle.push_back(abc);
        }

        pso::gbest_fit = 1.7976931348623158e+308;

}
```

```
/*
void pso::setDefaultOrientation(qt defaultOrientation,int i)
{

                particle[i].current_orientation = defaultOrientation;

}
void pso::setDefaultTorsion(fl defaultTorsion,int i,sz which)
{

                particle[i].current_torsion[which] = defaultTorsion;

}
*/

void pso::updateVelocity(rng& generator,int i)
{
                double r1 = random_double(this->R1Min_,this->R1Max_,generator);
                double r2 = random_double(this->R2Min_,this->R2Max_,generator);

                //std::cout<< "w:" << w <<"       c1:"<<c1 << "          c2:"<<c2<<"\n";
                //std::cout << " before compute new:" << particle[i].velocity[0] << ',' << particle[i].velocity[1]
<< ','<< particle[i].velocity[2] << '\n';
                //std::cout << "1:" << particle[i].velocity[1] << '*' << w << '+' << c1 << "*(" <<
particle[i].pbest_pos[1]<< '-' <<particle[i].current_position[1]<< ")+" << c2 <<"*("<<pso::gbest_position[1]<<'-
'<<particle[i].pbest_pos[1]<<')'<<'\n';
                particle[i].velocity[0] = particle[i].velocity[0]*w+c1*r1*(particle[i].pbest_pos[0]-
particle[i].current_position[0])+c2*r2*(pso::gbest_position[0]-particle[i].pbest_pos[0]);
                particle[i].velocity[1] = particle[i].velocity[1]*w+c1*r1*(particle[i].pbest_pos[1]-
particle[i].current_position[1])+c2*r2*(pso::gbest_position[1]-particle[i].pbest_pos[1]);
                particle[i].velocity[2] = particle[i].velocity[2]*w+c1*r1*(particle[i].pbest_pos[2]-
particle[i].current_position[2])+c2*r2*(pso::gbest_position[2]-particle[i].pbest_pos[2]);
                //particle[i].velocity = k*(particle[i].velocity+c1*r1*(particle[i].pbest_pos-
particle[i].current_position) + c2*r2*(pso::gbest_position-particle[i].pbest_pos));
                //std::cout << " after compute new:" << particle[i].velocity[0] << ',' << particle[i].velocity[1] <<
','<< particle[i].velocity[2] << '\n';
```

```
}

void pso::updateVelocityO(rng& generator,int i)

{


        qt p1 = particle[i].pbest_orientation-particle[i].current_orientation;

        qt p2 = pso::gbest_orientation-particle[i].pbest_orientation;

                double r1 = random_double(this->R1Min_,this->R1Max_,generator);

                double r2 = random_double(this->R2Min_,this->R2Max_,generator);

                //double psi = c1+c2;

                //double k = 2 / (std::abs(2-psi-(std::sqrt(psi*psi-4))));


                particle[i].vO = particle[i].vO*this->w+c1*r1*quaternion_to_angle(p1)+c2*r2*quaternion_to_angle(p2);

                //particle[i].vO = k*(particle[i].vO+c1*r1*quaternion_to_angle(p1) + c2*r2*quaternion_to_angle(p2));

}

void pso::updateVelocityT(rng& generator,int i,sz which)

{


                double r1 = random_double(this->R1Min_,this->R1Max_,generator);

                double r2 = random_double(this->R2Min_,this->R2Max_,generator);

                //double psi = c1+c2;

                //double k = 2 / (std::abs(2-psi-(std::sqrt(psi*psi-4))));

                //std::cout<<"old velocity:" << particle[i].vT[which]<<"\n";

                //std::cout<<"pbest_torsion:" << particle[i].pbest_torsion[which] <<" current_torsion:" << particle[i].current_torsion[which] << " gbest_torsion:" << pso::gbest_torsion[which] << '\n';

                //std::cout<<"p1:" << this->c1*(particle[i].pbest_torsion[which]-particle[i].current_torsion[which]) << " p2:" << this->c2*(pso::gbest_torsion[which]-particle[i].pbest_torsion[which]) <<"\n";

                particle[i].vT[which] = particle[i].vT[which]*this->w+c1*r1*(particle[i].pbest_torsion[which]-particle[i].current_torsion[which])+c2*r2*(pso::gbest_torsion[which]-particle[i].pbest_torsion[which]);

                //particle[i].vT[which] = k*(particle[i].vT[which]+c1*r1*(particle[i].pbest_torsion[which]-particle[i].current_torsion[which]) + c2*r2*(pso::gbest_torsion[which]-particle[i].pbest_torsion[which]));


                //std::cout<<"new velocity:" << particle[i].vT[which]<<"\n";
```

```
}

void pso::computeNewPositions(int i)

{

                //std::cout << " before compute new:" << particle[i].current_position[0] <<'+'<< particle[i].velocity[0] << ',' << particle[i].current_position[1] <<'+'<< particle[i].velocity[1]<< ','<< particle[i].current_position[2] <<'+'<< particle[i].velocity[2]<< '\n';

                particle[i].current_position[0] = particle[i].current_position[0] + particle[i].velocity[0];

                particle[i].current_position[1] = particle[i].current_position[1] + particle[i].velocity[1];

                particle[i].current_position[2] = particle[i].current_position[2] + particle[i].velocity[2];

                //std::cout << " after compute new:" << particle[i].current_position[0] << ',' << particle[i].current_position[1] << ','<< particle[i].current_position[2] << '\n';



                if(particle[i].current_position[0] < corner1[0])

                        particle[i].current_position = random_in_box(this->corner1,this->corner2,this->g);

                if(particle[i].current_position[1] < corner1[1])

                        particle[i].current_position = random_in_box(this->corner1,this->corner2,this->g);

                if(particle[i].current_position[2] < corner1[2])

                        particle[i].current_position = random_in_box(this->corner1,this->corner2,this->g);


                if(particle[i].current_position[0] > corner2[0])

                        particle[i].current_position = random_in_box(this->corner1,this->corner2,this->g);

                if(particle[i].current_position[1] > corner2[1])

                        particle[i].current_position = random_in_box(this->corner1,this->corner2,this->g);

                if(particle[i].current_position[2] > corner2[2])

                        particle[i].current_position = random_in_box(this->corner1,this->corner2,this->g);


}

//void pso::computeNewOrientation(void(*increment)(qt&,const vec&),int i)

void pso::computeNewOrientation(int i)

{
```

```
                vec tmp_v = particle[i].vO;

                //std::cout<< " before compute new:" << particle[i].current_orientation.R_component_1() << ',' << particle[i].current_orientation.R_component_2() << ','<< particle[i].current_orientation.R_component_3() <<','<< particle[i].current_orientation.R_component_4()<< "\n";

                quaternion_increment(particle[i].current_orientation, tmp_v);

                //std::cout<< " after compute new:" << particle[i].current_orientation.R_component_1() << ',' << particle[i].current_orientation.R_component_2() << ','<< particle[i].current_orientation.R_component_3()<<','<< particle[i].current_orientation.R_component_4()<< "\n";

        }

        void pso::computeNewTorsion(int i,rng& generator,sz which)

        {

                //std::cout<< " before compute new:" << particle[i].current_torsion[which] << "velocity:" << particle[i].vT[which]<<"\n";

                particle[i].current_torsion[which] = particle[i].current_torsion[which] + particle[i].vT[which];

                //std::cout<< " after compute new:" << particle[i].current_torsion[which] << "\n";

        //particle[i].current_torsion = random_fl(-pi, pi, generator);


                if(particle[i].current_torsion[which] > pi)

                        particle[i].current_torsion[which] = random_fl(-pi, pi, this->g);

                else if(particle[i].current_torsion[which] < -pi)

                        particle[i].current_torsion[which] = random_fl(-pi, pi, this->g);

        }



        void pso::updatePersonalBest(int i,double e)

        {

                particle[i].pbest_fit = e;

                //particle[i].pbest_pos = particle[i].current_position;

        }

        /*

        void pso::updatePersonalBestO(int i,double e)

        {

                //particle[i].pbest_fitForO = e;

                particle[i].pbest_fit = e;

                //particle[i].pbest_orientation = particle[i].current_orientation;
```

```
}
void pso::updatePersonalBestT(int i,double e,sz which)
{
	//particle[i].pbest_fitForT[which] = e;
	particle[i].pbest_fit = e;
	//particle[i].pbest_torsion[which] = particle[i].current_torsion[which];
}*/

void pso::updateGlobalBest(int i)
{

	pso::gbest_fit = particle[i].pbest_fit;
	//pso::gbest_position = particle[i].pbest_pos;

}
/*
void pso::updateGlobalBestO(int i)
{
	//pso::gbest_fitForO = particle[i].pbest_fitForO;
	pso::gbest_fit = particle[i].pbest_fit;
	//pso::gbest_orientation = particle[i].pbest_orientation;
}
void pso::updateGlobalBestT(int i,sz which)
{
	//pso::gbest_fitForT[which] = particle[i].pbest_fitForT[which];
	pso::gbest_fit = particle[i].pbest_fit;
	//pso::gbest_torsion[which] = particle[i].pbest_torsion[which];
}
*/

double pso::getPersonalBest(int i)
{
```

```
		return particle[i].pbest_fit;
	}
/*
	double pso::getTmpFit(int i)
	{
		return particle[i].tmp_fit;
	}
	void pso::setTmpFit(int i,double e)
	{
		particle[i].tmp_fit = e;
	}

	double pso::getPersonalBestO(int i)
	{
		//return particle[i].pbest_fitForO;
		return particle[i].pbest_fit;
	}
	double pso::getPersonalBestT(int i,sz which)
	{
		//return particle[i].pbest_fitForT[which];
		return particle[i].pbest_fit;
	}
	*/

	void pso::updateBestPosition(int i,vec pos)
	{
		particle[i].pbest_pos = pos;

	}
	void pso::updateBestOrientation(int i, qt orientation)
	{
		particle[i].pbest_orientation = orientation;
```

```
}
void pso::updateBestTorsion(int i, fl torsion,sz which)
{
	particle[i].pbest_torsion[which] = torsion;
}

/*
	void pso::setCurrentPosition(int i ,vec pos)
	{
		particle[i].current_position = pos;
	}
	void pso::setCurrentOrientation(int i ,qt orientation)
	{
		particle[i].current_orientation = orientation;
	}
	void pso::setCurrentTorsion(int i ,fl torsion, sz which)
	{
		particle[i].current_torsion[which] = torsion;
	}
	*/

	vec pso::getCurrentPosition(int i)
	{
		return particle[i].current_position;
	}

	qt pso::getCurrentOrientation(int i)
	{
		return particle[i].current_orientation;
	}
	fl pso::getCurrentTorsion(int i,sz which)
	{
```

```
                return particle[i].current_torsion[which];
        }



        void pso::print()
        {
                for(int i=0;i<this->number;i++)
                {
                        std::cout << "========================bird" << i << "========================" << '\n';
                        std::cout << "current_position:" << particle[i].current_position[0]<< ',' << particle[i].current_position[1] << ',' << particle[i].current_position[2] << '\n';
                        std::cout << "pbest_position:" << particle[i].pbest_pos[0] <<',' << particle[i].pbest_pos[1] << ','<< particle[i].pbest_pos[2] << '\n';
                        std::cout << "gbest_position:" << pso::gbest_position[0] <<',' << pso::gbest_position[1] << ','<< pso::gbest_position[2] << '\n';

                        std::cout << '\n';
                        std::cout << "current_orientation:" << particle[i].current_orientation <<'\n';
                        std::cout << "pbest_orientation:" << particle[i].pbest_orientation <<'\n';
                        std::cout << "gbest_orientation:" << pso::gbest_orientation <<'\n';
                        //std::cout << "pbest_fitO:" << particle[i].pbest_fitForO <<'\n';
                        std::cout << '\n';

                        std::cout << "pbest_fit:" << particle[i].pbest_fit <<'\n';
                        std::cout << "gbest_fit:" << pso::gbest_fit <<'\n';

                }
        }
```

10.3.4 mutate.h

```
/*

  Copyright (c) 2006-2010, The Scripps Research Institute

  Licensed under the Apache License, Version 2.0 (the "License");
```

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

Author: Dr. Oleg Trott <ot14@columbia.edu>,

The Olson Lab,

The Scripps Research Institute

*/

```
#ifndef VINA_MUTATE_H

#define VINA_MUTATE_H

#include "pso.h"

#include "model.h"

#include "quasi_newton.h"

// does not set model

//void mutate_conf(output_type& c, const model& m, fl amplitude, rng& generator,pso*,const precalculate&,const igrid&,change&,const vec&,quasi_newton&,int);

void mutate_conf(output_type& c, const model& m, fl amplitude, rng& generator,pso*,const precalculate&,const igrid&,change&,const vec&,quasi_newton&,int);

void mutate_conf(conf& c, const model& m, fl amplitude, rng& generator);

//void mutate_conf(output_type& cand, const model& m, fl amplitude, rng& generator, precalculate& p,igrid& ig,change& g,pso ps);
```

```
#endif
```

10.3.5 monte_carlo.h

```
/*

   Copyright (c) 2006-2010, The Scripps Research Institute

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.

   Author: Dr. Oleg Trott <ot14@columbia.edu>,
           The Olson Lab,
           The Scripps Research Institute

*/

#ifndef VINA_MONTE_CARLO_H
#define VINA_MONTE_CARLO_H

#include "ssd.h"
#include "incrementable.h"
#include "pso.h"

struct monte_carlo {
```

```
	unsigned num_steps;

	fl temperature;

	vec hunt_cap;

	fl min_rmsd;

	sz num_saved_mins;

	fl mutation_amplitude;

	ssd ssd_par;

	monte_carlo() : num_steps(2500), temperature(1.2), hunt_cap(10, 1.5, 10), min_rmsd(0.5), num_saved_mins(50), mutation_amplitude(2) {} // T = 600K, R = 2cal/(K*mol) -> temperature = RT = 1.2;  num_steps = 50*lig_atoms = 2500



	output_type operator()(model& m, const precalculate& p, const igrid& ig, const precalculate& p_widened, const igrid& ig_widened, const vec& corner1, const vec& corner2, incrementable* increment_me, rng& generator) const;

	output_type many_runs(model& m, const precalculate& p, const igrid& ig, const vec& corner1, const vec& corner2, sz num_runs, rng& generator) const;



	void single_run(model& m, output_type& out, const precalculate& p, const igrid& ig, rng& generator) const;

	// out is sorted

	void operator()(model& m, output_container& out, const precalculate& p, const igrid& ig, const precalculate& p_widened, const igrid& ig_widened, const vec& corner1, const vec& corner2, incrementable* increment_me, rng& generator) const;

	void many_runs(model& m, output_container& out, const precalculate& p, const igrid& ig, const vec& corner1, const vec& corner2, sz num_runs, rng& generator) const;



};



#endif
```

10.3.6 pso.h

```
#ifndef PSO_H_

#define PSO_H_



#include "common.h"

#include "conf.h"

#include <vector>
```

```
class pso
{
public:
	struct bird{
		vec velocity,vO;  /* current velocity */

		vec pbest_pos;
		vec current_position;

		qt pbest_orientation; /* particle the best position  */
		qt current_orientation;

		fl* pbest_torsion;
		fl* current_torsion;
		fl* vT;

		//double* pbest_fitForT;
		double pbest_fit,tmp_fit;  /*particle personal best value*/
		//double pbest_fitForO;
	};

	int torsionSize;
	double w,c1,c2;
	rng g;

	int number;
	vec corner1,corner2;
	static vec gbest_position;
	static qt gbest_orientation;
	static fl* gbest_torsion;
	static double gbest_fit;
```

```
        //static double gbest_fitForO;
        //static double* gbest_fitForT;
        std::vector<bird> particle;


         double R1Max_;
         double R1Min_;
         double R2Max_;
         double R2Min_;


        pso(int,const vec,const vec,rng&,conf&);
        void init(rng&,conf&);

/*Update Velocity for vector*/
void updateVelocity(rng&,int);
void updateVelocityO(rng&,int);
void updateVelocityT(rng&,int,sz);


/*Compute the new pose*/
void computeNewPositions(int);
//void computeNewOrientation(void(*)(qt&,const vec&),int);
        void computeNewOrientation(int);
        void computeNewTorsion(int,rng&,sz);


/*Update personal best value*/
        void updatePersonalBest(int,double);


/*Update global best value*/
```

```
        void updateGlobalBest(int);


  /*Get personal best value*/
        double getPersonalBest(int);       //return the personal best value


  /*Set personal best vector*/
  void updateBestPosition(int,vec);
  void updateBestOrientation(int,qt);
  void updateBestTorsion(int,fl,sz);


  /*Set current vector*/
 // void setCurrentPosition(int,vec);
        //void setCurrentOrientation(int,qt);
        //void setCurrentTorsion(int,fl,sz);



  /*Get current vector*/
        vec getCurrentPosition(int);
  qt getCurrentOrientation(int);
  fl getCurrentTorsion(int,sz);


  /*Print out the birds information*/
        void print();



};



#endif /*PSO_H_*/
```