การลดปริภูมิของการค้นหาสำหรับการคำนวณการเข้าจับเชิงโมเลกุล

นายทศพล บวรธนิตกุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรดุษฎีบัณฑิต
สาขาวิชาวิทยาศาสตร์นาโนและเทคโนโลยี (สหสาขาวิชา)
บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2555
ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

SEARCH SPACE REDUCTION TECHNIQUE FOR MOLECULAR DOCKING CALCULATION

Mr. Thotsaphon Boworndhanitkun

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy Program in Nanoscience and Technology
(Interdisciplinary Program)
Graduate School
Chulalongkorn University
Academic Year 2012

Thesis Title	SEARCH SPACE REDUCTION TECHNIQUE FOR MOLECULAR DOCKING CALCULATION
By	Mr. Thotsaphon Bowornthanitkun
Field of Study	Nanoscience and Technology
Thesis Advisor	Assistant Professor Somsak Pianwanit, Ph.D.

Accepted by the Graduate School, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

ทศพล บวรธนิตกุล : การลดปริภูมิของการค้นหาสำหรับการคำนวณการเข้าจับเชิงโมเลกุล (SEARCH SPACE REDUCTION TECHNIQUE FOR MOLECULAR DOCKING CALCULATION) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร.สมศักดิ์ เพียรวณิช, 112 หน้า.

งานวิจัยชิ้นนี้ได้เสนอการใช้เทคนิคแบ่งแยกและพิชิตเพื่อที่จะแก้ปัญหาเกี่ยวกับการจับกันของตัวยาที่เป็นสารอินทรีย์โมเลกุลเล็ก ๆ (ligand) กับโปรตีน เทคนิคนี้สามารถนำไปใช้ในกระบวนการคัดกรองสารเพื่อทำการออกแบบตัวยาได้ ขนาดของพื้นที่ของพื้นผิวพลังงานซึ่งมีจำนวนมากสามารถถูกลดทอนให้เหลือแค่ส่วนที่จำเป็นต้องทำการค้นหาจริง ๆ ได้ด้วยการใช้วิธีการทางการจัดเรียงลำดับทางคอมพิวเตอร์ ส่วนการเข้าถึงข้อมูลของแต่ละจุดพลังงานจะทำโดยอาศัยการใช้ดัชนีค้นหาร่วมกับการใช้กลไกของภาษาเชิงวัตถุร่วมกัน การใช้ดัชนีค้นหาจะสามารถทำให้การเข้าถึงแต่ละตำแหน่งได้ในเวลาอันสั้นและเท่ากันในทุกจุด ส่วนภาษาเชิงวัตถุมีความสามารถในการซ่อนข้อมูลและรวบรวมคลาสต่าง ๆ เข้าด้วยกันได้ สำหรับการจัดเรียงข้อมูลจะใช้วิธีการเรียงข้อมูลที่ไม่ขึ้นกับความแปรปรวนของข้อมูลเข้าโดยใช้การเรียงแบบ heap ซึ่งเป็นการเรียงข้อมูลที่มีประสิทธิภาพสูงมาก การใช้วิธีการต่าง ๆ ดังที่กล่าวมาข้างต้นจะสามารถทำให้การค้นหาพื้นผิวพลังงานที่มีขนาดใหญ่แต่มีความจำเพาะสูงสามารถทำได้ครอบคลุม การนำวิธีการสร้างภาพทางคอมพิวเตอร์มาใช้ในการสร้างโครงสร้างของ ligand จะทำให้การทำงานของโปรแกรมนี้สามารถทำงานแบบขนานกันได้อย่างมีประสิทธิภาพสูง งานวิจัยฉบับนี้ได้นำเสนอวิธีการคัดกรองยาแบบใหม่โดยใช้ค่าพลังงานที่ได้มาจากแผนที่พลังงานของโปรแกรม Autodock โดยที่ตัวโปรแกรมจะให้คำตอบออกมาเป็นตำแหน่งและทิศทางที่ ligand สามารถวางตัวอยู่ได้ในโปรตีน อย่างไรก็ตามมุมบางมุมและพื้นที่บางพื้นที่ที่ไม่ได้อยู่บนจุดพลังงานอาจไม่ได้ถูกค้นหา ซึ่งก็สามารถแก้ไขได้โดยการนำเอาผลลัพธ์ที่ได้มาค้นหาในบริเวณใกล้เคียงกันซ้ำอีก

สาขาวิชา วิทยาศาสตร์นาโนและเทคโนโลยี	ลายมือชื่อนิสิต
ปีการศึกษา 2555	ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก

5087768120 : MAJOR : NANOSCIENCE AND TECHNOLOGY
KEYWORDS : MOLECULAR DOCKING, ALGORITHM, COMPUTER GRAPHIC

THOTSAPHON BOWORNTHANITKUN : SEARCH SPACE REDUCTION TECHNIQUE FOR MOLECULAR DOCKING CALCULATION.
ADVISOR : ASST. PROF. SOMSAK PIANWANIT, Ph.D., 112 pp.

To dock organic ligands in to protein binding sides by using devide-and-conqure method. The method can be used in the virtual screening process of designing a specific protein ligand. The molecular surface energy representation of protein and ligand can be used to filter out by sorting algorithm along with indexing method. Object oriented paradigm, a powerful tool for data abstraction and encapsulation, can be used for accessing all index in real time. Sorting algorithm by heap sort can guarantee the fastest and time consistency regardless of the variation of input data. Combining all of these can make sure that all possible data can be search in the complete and optimum way. Computer graphic calculation is a really good in parallel computing. Introducing computer graphic calculation can guarantee parallelizable in the most effective way. This paper will introduce the new method by using energy based virtual screening to do the complete search of autodock grid map energy. The result of program is a position and orientation of ligand in enzyme. However, there are some missing angle and interpolation area between the grid map corners. This problem can be solved by doing recursive search in the near area of the first output data.

Field of Study : Nanoscience and Technology Student's Signature ....

Academic Year : 2012 Advisor's Signature ....

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation and sincere gratitude to my advisor, Assistant Professor Dr. Somsak Pianwanit for his helpful advises guidance, encouragement and constructive criticism throughout my research study which enable me to accomplish this thesis.

I also would like to thank all thesis committee including, Associate Professor Dr. Vudhichai Parasuk, Assistant Professor Dr. Rachada Kongkachandra, Dr. Ratthapol Rangupan, Assistant Professor Dr. Warangkana Warisnoicharoen and Assistant Professor Dr. Sukree Sinthupinyo for their useful comments.

This study was supported partly by a grant from the Ministry of University Affairs, Thailand and the Graduate School, Chulalongkorn University.

Finally, I would like to express my deepest appreciation to my family and friends for their encouragement and morale that have made me to complete this work.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

1.1 Drug Development Process

There is an increasing rate of emerging infectious diseases nowadays. Most of them are caused by some kind of microorganisms, including both new born virus and bacteria, as well as those evolve themselves reisistant to conventional antimicrobial drugs. These living organisms have biological activities to produce biochemical compounds, such as enzymes or some types of RNA, for serving their pathway of life. These compounds, especially enazymes, are the main targets of designed drugs in order to kill the infectious microorganisms efficiently. Thus, development of a drug that can bind specifically to the target enzyme, or named as "receptor", to inhibit its biological function is necessary for a treatment of the disease. However, in general, research and development for a single drug is extremely cost- and time-consuming. In average, it takes around 10-15 years and costs about $1.8 billion to take a new compound to market[1].

A drug development process usually starts with finding lead compounds, which can be done in several ways. A simple method is to screen biological activities (*in vitro*) of all available compounds in hand, which can be as much as hundred thousands compounds. This is a kind of "trial and error" method and is not effective at all. Then, the obtained lead compouds must be studied carefully in animal (*in vivo*) to investigate their pharmacodynamics and pharmokinetics as well as toxicity. Most of lead compounds fail in this step. Therefore, only few numbers of lead compounds can pass to the next step, which is testing in human or clinical research. This step is by far the logest protion of the drug development process and can takes from 2 to 10 years[1].

Considering the drug development process, it can be seen that finding appropriate lead compounds is very important since it can cut down both time and budget. With a remarkable progress of computer technology, many computational techniques have been invented to facilitate the finding of appropriate lead compounds, for example, quantitative structure-activity relationship (QSAR)[2] both classical and three dimensional (3D) QSAR[3-4], molecular docking[5], pharmacophore searching[6], and virtual screening[7].

1.2 Molecular Docking

Molecular docking is a computational technique to predict a binding between two molecules; in general a larger molecule is called receptor while a smaller molecule is called ligand. Nowadays, it is widely accepted and has been using worldwide as a structure-based drug design tool to study enzyme-ligand interaction[8-11]. The name docking comes from the word “dock”. Dock is a place in a port where ships are loaded, unloaded, or repaired. In molecular docking, ligand is analogous to a ship and receptor is analogous to a dock. The main goal of molecular docking is to get the binding conformation regardless of any method, as long as that method is a legitimate scientific method. Molecular docking has been used in several applications in drug design and discovery process[12], for example, structure-activity studies, lead optimization (by providing guideline how to modify chemical structure to maximize binding interaction), virtual screening to find potential leads, X-ray crystallography to assist a fitting of substrates and inhibitors to electron density, chemical mechanism studies, and combinatorial library design.

The inputs of molecular docking are a 3D structure of receptor; enzyme or protein, and a 3D structure of ligand, usually a small chemical compound. The 3D structure of receptor can be obtained by either experiment, e.g. the Protein Data Bank database[13], or theory, i.e. using molecular modeling technique to build a model structure. Similarly, 3D structure of ligand can either be taken directly from experimental data or be designed by chemist, using any molecular modelling software capable to create 3D structure.

The aim of docking calculation is to find the best configuration (in terms of both position and orientation) of ligand within receptor structure. In principle, docking calculation is comprised of two interrelated steps; first by searching or sampling configurations of the ligand in the active site of the protein (which is usually referred to as pose); then ranking these conformations via a scoring function. Ideally, configurations generated from the searching or sampling algorithm must include the experimental binding mode and the scoring function should be able to pick it out by ranking it with the highest score. Several excellent reviews of molecular docking methods[11,14-16] as well as publications comparing the performance of various molecular docking tools[17-22] are available.

1.2.1 Searching Algorithm

There are many searching methodologies applied to molecular docking[11,15], for example, matching algorithm[23], incremental construction method[24], and stochastic methods[25-27], in which the latter methods belong to artificial intelligence (AI) techniques. Usually, searching problem in AI can be depicted as an agent or agents working in an environment. The agent can have perception in the environment and take some set of actions based on defined set of algorithms inside itself. A goal of action is to make the agent change its state in order to achieve the desired goal. In rigid docking, the desired goal is easily describe as a position in receptor and the orientation of ligand binding with each other in order to get the higest fitness score. The agent can be replaced by ligand and the environment is all possible binding sites in the protein (search space). According to this idea, three searching methodologies in AI known as Monte Carlo simulated annealing, genetic algorithm (GA) and Lamarckian genetic algorithm, are applied in molecular docking calculations[25-27]. The next paragraph will explain two of these searching methodologies in brief.

Monte Carlo simulated annealing: this methodology come from the name Monte Carlo and simulated annealing combined together. Monte Carlo is widely known for its casino, and its gambling. So this methodology is based upon randomness. Simulated annealing term came from engineering field. Anneal means to heat metal or glass and allow it to cool slowly, in order to make it harder. This algorithm uses the idea of thermodynamic in order to escape from local minima point and give more probability to achieve global minima point. Nevertheless, Monte Carlo simulated annealing is considered an obsolete algorithm because it is very slow and always stuck in local minima.

Genetic algorithm: this methodology is trying to mimic the natural reproduction way. The basic idea of genetic is to get some set of genes from both parents and inherit them to their children. Gene is a set of string that encoding the orientation and position of ligand in receptor space. Start with generating a lot of ligand’s position and orientation randomly. Then match all of them in couple in random manner. The children from each couple will receive a new position and orientation based on parent’s gene. The children that have lower energy are going to be accepted, leave out most of the higher energy children. After that, doing the cross over again until it reaches a number of setting iteration (or usually known as generation), However, sometime all parents might have a set of bad gene. This will lead to the local minima problem, where children have no way to achieve a lower energy even if there is. To escape from local minima problem, genetic algorithm use the idea of mutation, again mimic from the nature. Mutation is a set of instruction that randomly change some bit, or string in chromosome. Since the orientation and position in receptor space are encoded into a set of string in chromosome, then the change in this string can let the children escape to the new position and orientation. Genetic algorithm can change to evolution algorithm, if the crossover rate is set to be zero, meaning to have only mutation in the system. According to this, the number of mutation should be considered and adjusted by user, even though the software might already set the rate of mutation randomly. Genetic algorithm can give a better result based on the number of parents and the number of iterations and it does not guarantee whether the output will be the same or not.

1.2.2 Scoring Function

Apart from the difficulty in searching algorithm, a scoring function is another significant trouble. The purpose of the scoring function is to identify the correct poses from ensemble of poses generated by searching/sampling algorithm. There are various scoring functions which employ different formulars depending on the theory or concept of that scoring. Scoring function can be divided into force-field based, empirical and knowledge-basee scoring functions[8]. Each function has its own advantage and disadvantage. There is no common clue which scoring function is the best for a given system. Therefore, development of new scoring function is an active research area in molecular docking field although it is quite difficult and time-consuming task.

Currently, there are several molecular docking softwares available both freeware, e.g. AutoDock[26], DOCK[27], ZDOCK[28], and commercial software, for example, GOLD[29], Glide[30], FlexX[24]. Among these, the AutoDock is a very popular and widely used program due to its efficiency and free availability, which includes a source code of the program. Therefore, the AutoDock is choosen for our research.

1.3 Objective

Although the AutoDock is very efficient, it has some drawbacks. The searching/sampling algorithm implemented in the AutoDock is genetic algorithm, which is a stochastic technique. It is well known that a major drawback of the stochastic technique is repeatability since it is based on statistics and randomness. Another drawback of the AutoDock is a huge search space, i.e. it includes all 3D space within a user defined box covering a binding site of receptor. Combining these two drawbacks, considerable calculation time is probably devoted to generation of enormous unsuitable configurations randomly from a huge search space, which results in a long computational time. In order to solve the problems in both repeatability and efficiency in computational time, it is, therefore, the goal of this research work to develop a new search algorithm to reduce search space for molecular docking calculation using divide-and-conquer approach together with computer graphic algorithm. In addition, our program will be developed using JAVA language, which is independent of the computer platform, thus it can be run natively on a computer with any operating system.

CHAPTER II

METHODOLOGY

2.1 Docking procedure in AutoDock

The purpose of this research is to develop new searching algorithm to reduce search space which results in shorter computational time and ensure repeatability of the docking. However, the scoring function is still based on that of the AutoDock. Therefore, information about a docking procedure in the AutoDock is beneficial and facilitative for the understanding of our new algorithm. But first, it has to clarify that the AutoDock is actually a suite of programs consisting of autotor, autogrid, and autodock sub-programs, in which each sub-program has its own function. Here are steps for performing docking calculation using the AutoDock[31].

1. Coordinate file preparation. Three dimensional structures of receptor and ligand are required. If the structure is obtained from X-ray crystallography, hydrogen atoms must be added. This process can be done using any molecular modeling software but it is very convenience with a use of the AutoDockTools[32], a separate graphical user interface designed for the AutoDock. Then, the AutoDockTools assigns atomic charges and atom types for all atoms and the molecule is saved in an extended PDB format, termed PDBQT.

2. Grid map calculations. Since the fitness score must be calculated for every configuration generated during the searching/sampling algorithm, in this case a genetic algorithm, the AutoDock employs a rapid grid-based evaluation technique to accelerate this process. The scoring function in the AutoDock is an extension of force-field-based scoring functions considering four terms: dispersion/repulsion interaction (or van der Waals intereaction using Lennard-Jones pair potential), electrostatic energy (using Coulomb potential), hydrogen bonding energy, and desolvation energy[33]. For the van der Waals intereaction, a grid map for each atom type found in the ligand is generated.

For example, if the ligand is methanol, CH_3OH, there are 3 atom types and grid maps for carbon, hydrogen, and oxygen atoms will be created. Details of this step, using carbon atom type as example, are explained below.

a. A three dimensional lattice of regularly spaced points (usually called grid box) is built and is placed on the receptor in such a way that it covers all of the binding site area as shown in Figure 1.

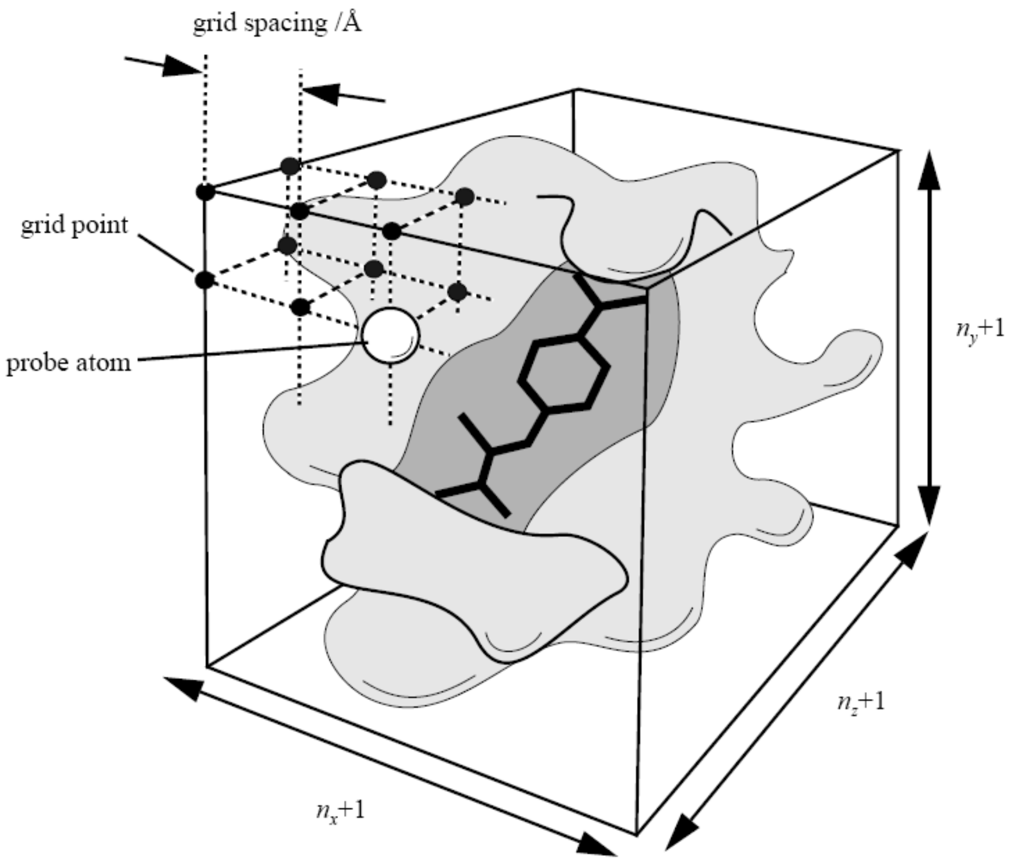


Figure 2.1 Grid map used for calculation in Autogrid[34].

b. A probe atom, in this case is a carbon atom, is placed at one grid point. The interaction energy between this carbon atom and the receptor is calculated and is assigned to the grid point. Then, the probe atom is moved to the next grid point and intereaction energy is calculated. This step is repeated until all the grid points are visited. Finally, a carbon grid map is obtained.

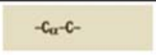
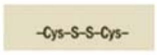
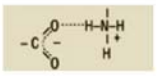
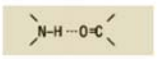
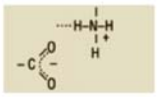
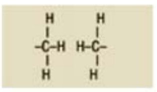
Van der Waals grid map for each atom types of ligand is prepared in the same way as above (step 2A and 2B). In addition, grid maps for electrostatic energy as well as desolvation energy are created using also the same procedure but a probe atom is just a point charge with +1 value for the electrostatic map and is a probe point with solvation parameter for the desolvation map.

3. Docking simulation. Docking calculation is carried out with Autodock program using the chosen search/sampling algorithm, typically the Lamarckian genetic algorithm (LGA). During the search, the fitness score is evaluated by reading van der Waals interaction, electrostatic energy, hydrogen bonding energy, and desolvation energy for each atom of ligand from the corresponding grid map, and then summing all energy terms of all atoms together. Since the LGA is based on statistics and random, the Autodock is run several times to give several docked conformations, and analysis of the predicted energy and the consistency of results is combined to identify the best solution. According to this procedure, the search space is enormous because every grid points of all grip maps are included. It is impossible to search for every possibility thus the obtained docked structure may not match the experiemental data.
4. Analysis step. The AutoDockTools can be used to analyse the results, in which interactions between ligand and receptor is usually visualized. Hydrogen intereaction is also another important property for analysis.

2.2 Concept of our algorithm

Binding between enzyme and ligand is very specific. It means binding site of enzyme is not a huge area as compare to the size of ligand. When ligand binds to the enzyme, ligand has to stay at the right position and orientation. The right position can be determined by using the knowledge of physical chemistry, for example, charge-charge interaction. Since most of the enzymes are protein, all forces that can hold ligand in the right position have to come from protein structure (see Table1).

Table 1.1 Chemical interaction that stabilizes polypeptides.

Interaction	Example	Distance dependence	Typical distance	Free energy (bond dislocation enthalpies for the covalent bond)
Covalent bond		-	1.5 Å	356 kJ/mole (610 kJ/mole for a C=C bond)
Disulfide bond		-	2.2 Å	167 kJ/mole
Salt bridge		Donor (N) and Acceptor (O) atoms < 3.5 Å	2.8 Å	12.5-17 kJ/mole; may be as high as 30 kJ/mole for fully or partially buried salt bridges. Less if salt bridge is external.
Hydrogen bond		Donor (N) and Acceptor (O) atoms < 3.5 Å	3.0 Å	2-6 kJ/mole in water; 12.5-21 kJ/mole if either donor or acceptor is charged.
Long-range electrostatic		Depends on dielectiric constant of medium. Screened by water. 1/r dependence	Variable	Depends on distance and environment. Can be very strong in nonpolar region but very weak in water.
Van der Waals		Short range. Fall off rapidly beyond 4 Å separation. $1/r^6$ dependence	3.5 Å	4 kJ/mole (4-17 in protein interior) depending on the size of group (for comparison, the average thermal energy of molecules at room temperature is 2.5 kJ/mole)

In Table 1, molecular docking considers only weak forces which are van der Waals interaction, long-range electrostatic interaction, hydrogen bond, and salt bridge. Even though some enzyme-ligand complexes are binding with covalent bond, molecular docking will not count them in. In molecular docking, all these weak forces will add up together for holding ligand. Since energy is the integration of all forces through the distance. The lowest energy point or area is the most favorite place of ligand to stay. As stated in the previous section that considering all grid points of every grid map in GA search algorithm as in the AutoDock results in a tremendous search space, it is thus our goal to develop a new algorithm that can significantly reduce a search space in molecular docking. Comparing the free energy among these interaction types (see Table 1), it is possible to use only the van der Waals interaction to screen out some part of the search space (grid points) that have higher energy than a defined threshold, thus, shorten calculation time. Therefore, this hypothesis is used to develop our new algorithm using a divide-and-conquer approach together with a computer graphic algorithm. Here are details of steps in our algorithm.

1. Coordinate file preparation.
2. Grid map calculations.

Step 1 and 2 are exactly the same as those in the AutoDock procedure because the scoring function of AutoDock is used. Only searching algorithm is modified from the genetic algorithm to our new algorithm.

3. Define the threshold for van der Waals interaction energy. Grid point with energy higher than this threshold is defined as high energy and will not be considered (exclude from the search space).
4. Find the lowest redundant atom founded in ligand. This atom is called “rst”. The lowest redundant atom refers to an atom type of ligand with the minimum number of atom. For example, if a ligand is hydroquinone, $C_6H_6O_2$, oxygen atom type has the minimum number of atom and is defined as the “rst” atom.
5. Sort energy in the “rst” grid map type, e.g. oxygen grid map.
6. Start from the first “rst” in ligand, put it at the lowest energy grid point of the “rst” grid map.
7. Find the longest intramolecular distance between “rst” and atom inside the ligand, and then create a vector connecting the two atoms (see Figure 2).

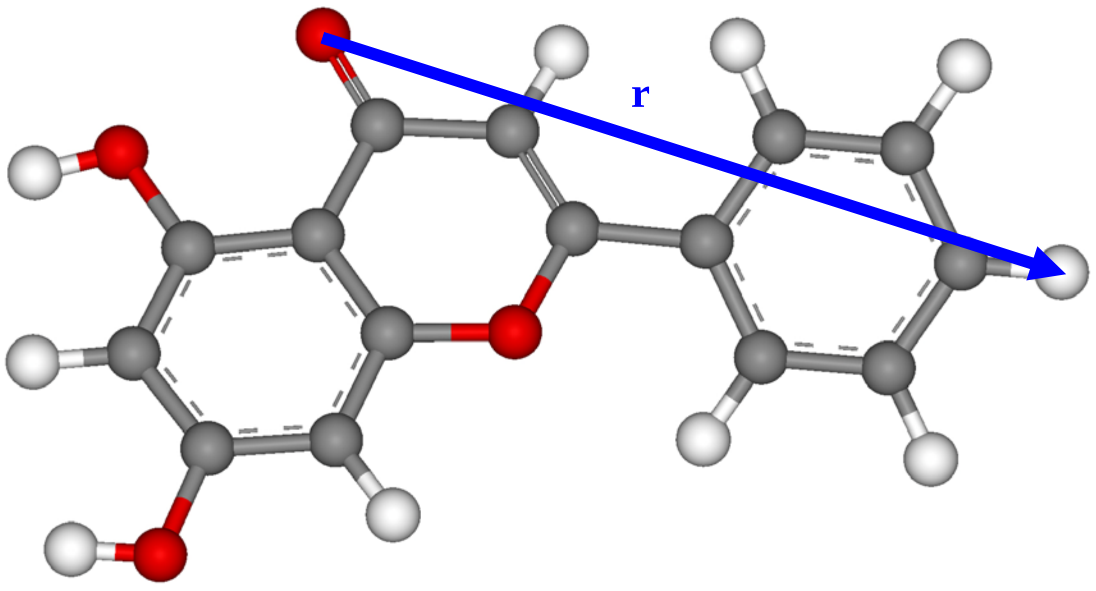


Figure 2.2 Defining a vector connecting between a central atom and the farthest atom.

8. Rotate the vector spherically by varying zeta (θ) and phi (ϕ) with a step size of 1 degree; see Figure 3 for the definition of zeta and phi angles. This step generates a large number of configurations.

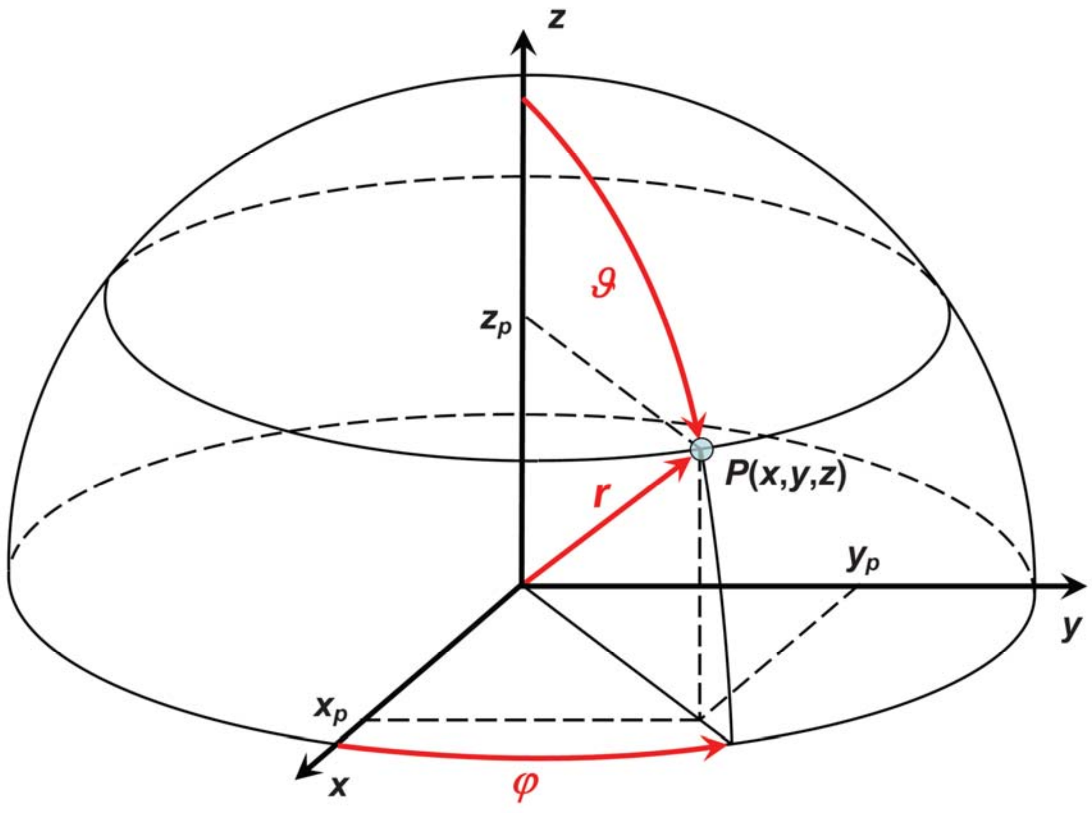


Figure 2.3 Spherical and Cartesian coordinates.

9. For each configuration generated in step 8, if the energy of the farthest atom is lower than the defined threshold, it is stored in the buffer of the program, otherwise, it is omitted.
10. Move the “rst” atom to the next lower energy grid point, repeat step 7 to 9.
11. Repeat step 10 for every grid points with energy lower than the threshold.
12. Change to the next “rst” atom of ligand. Repeat step 6 to 11 for all “rst” atoms.
13. Generate the output for visualization.

Interestingly, our algorithm uses computer graphic method in order to generate ligand at the grid position and orientation. The advantage of using computer graphic method is it can run in parallel very efficiently. In computer graphic, translation object in 3 dimension space has to deal with a lot of matrix multiplication. Row by Row matrix multiplication with buffer each row in the buffer array can reduce cache miss of the system. This yields to a very fast computing and searching the system. Rotation along ligand arbitrary matrix can be optimize by computing all rotational matrix first before multiple every ligand coordinate, represented in Cartesian coordinate. Gaussian elimination is used for doing the matrix inversion in order to reduce the time complexity of computing matrix transformation. These matrix codes can be implemented in java program.

For achieving the repeatability and hopefully to get as close as global minima in acceptable time. In this research work, the searching paradigm is changing from statistical based to rational search paradigm. Despite of random walk, the entire map should be sorted first before searching of grid points and angles begin. Sorting data in computer can be done in a very short period of time. To retrieve data with array index can be done in real time, regardless of the size of data. These two methods, sorting and indexing, combine together will enormously reduce the space and time that required when compared to brute force search. The chance of finding global minima should increase when compare to statistical based search as well.

Finally, all of the low energy point on sphere surface will be choose as a rotational point again but this time ligand will adjust its orientation vector to the minimum energy grid point of the farthest atom. This searching method will do iteratively until it reaches all possible grid points which have its energy under the threshold of its system. The ligands position, orientation and their energy will be kept in array until it reaches the maximum array index. Then the array will be sorted by heap sort algorithm. The next ligand position, orientation will compare its energy with the highest energy in array. If its energy is lower, then the array will keep the new ligand position, orientation and its energy by replacing the highest energy in array. After that, the array will be sorted again. Using this algorithm can guarantee that almost all possible ligand positions and orientations which have its own grid point energy under the threshold will be searched completely and optimally.

CHAPTER III

RESULTS AND DISCUSSION

3.1 Development of the new searching algorithm

According to the concept of our algorithm explained in Chapter 2, the JAVA program with a new searching algorithm for docking calculation were developed. The program consists of 8 classes, which are listed below.

1. MapMaker.java (source code is given in Appendix A). This class loads grid map generated from the AutoDock, keep them in the array and then sorts the energy from minimum to maximum values.
2. LSphereData.java (source code is given in Appendix B). This class finds the longest intramolecular distance between the “rst” atom and atoms inside the ligand. Create a vector and rotate it spherically. Then, position of the furthest atom of all points is obtained.
3. RetriveEnergy.java (source code is given in Appendix C). This is the core of our algorithm. It is used to put the “rst” atom into the lowest energy grid point and then to calculate the energy of the furthest atom for all configurations in the sphere. Determine whether the energy is higher than the threshold. Keep the acceptable point in the buffer.
4. Ligand.java (source code is given in Appendix D). This is used to store information about ligand.
5. EnCo.java (source code is given in Appendix E). This class encapsulates energy and coordinates together for the use of sorting purpose.
6. CUDockUtil.java (source code is given in Appendix F). It is a utility to read and write a pdb file.
7. MapConstant.java (source code is given in Appendix G). All the constants from the grid map are stored using this class.

8. MatrixUtil.java (source code is given in Appendix H). All operation of matrix is done with this class.

3.2 Efficiency of our new searching algorithm

In order to examine an efficiency of our new searching algorithm, the Astex diverse set[20] was choosen. The Astex diverse set is a standard test set for the validation of protein-ligand docking performance. It contains 85 high-quality X-ray complex structures taken from the Protein Data Bank. In this study, 15 ligand-enzyme complex structures were randomly selected from the Astex diver set. Details of these 15 complexes are given in Table 3.1. For each complex, a ligand is split up from an enzyme and then our new searching algorithm is used to dock it back. By comparing configuration of the ligand between the docked and the X-ray structure, efficiency of our new searching algorithm can be evaluated.

In addition to the concept described in Chapter 2, an extension of our algorithm was also implemented. Originally, after definding the vector connecting between the “rst” atom and the farthest atom inside ligand, it is rotated spherically and if the energy of the farthest atom is higher than the defined threshold, the whole sphere is omitted (step 9 in section 2.2, Chapter 2). This procedure is now called “truncate” search algorithm. However, in an extension version, all the shere is incluced even some of the energys of the farthest atom are higher than the defined threshold. Therefore, this is called “full” search algorithm. Both algorithms were applied to all 15 compounds in the test set. A comparison of computational time used for docking calculation of each compound between the AutoDock and our two algorithms were given in Table 3.2. Note that the AutoDock calculations used all default setting parameters except the GA run, which was changed to 100 to increase a probability of finding the right configuration.

Table 3.1 List of X-ray complex structures used as testing set.

PDB code	Enzyme	No. of residue	Resolution	Ref.
1GPK	Acetylcholinesterase	537	2.10	[35]
1HVY	Thymidylate Synthase	288	1.90	[36]
1IG3	Thiamin pyrophosphokinase	263	1.90	[37]
1JLA	HIV-1 Reverse Transcriptase	560	2.50	[38]
1L2S	beta-lactamase	358	1.94	[39]
1M2Z	Glucocorticoid receptor	257	2.50	[40]
1NAV	Hormone receptor alpha 1	263	2.50	[41]
1OQ5	Carbonic anhydrase II	259	1.50	[42]
1P62	Deoxycytidine kinase	263	1.90	[43]
1T46	C-kit tyrosine kinase	313	1.60	[44]
1TZ8	Transthyretin	127	1.85	[45]
1UNL	Cyclin-dependent Kinase 5	292	2.20	[46]
1X8X	Tyrosyl-tRNA synthetase	322	2.00	[47]
1Y6B	VEGFR2 Kinase	366	2.10	[48]
2BR1	CHK1 Kinase	297	2.00	[49]

Table 3.2 Comparison of computational docking time between the AutoDock and our new search algorithm.

System	No. of residue	Computional Time (min:sec)		
		AutoDock	Truncate search algorithm	Full search algorithm
1GPK	537	171:25	0:44	6:37
1HVY	288	130:08	0:13	7:31
1IG3	263	152:10	0:05	31:32
1JLA	560	395:34	0:08	32:40
1L2S	358	150:55	0:21	30:00
1M2Z	257	199:13	0:20	34:20
1NAV	263	215:37	0:05	31:15
1OQ5	259	252:18	0:10	30:40
1P62	263	123:24	0:09	26:38
1T46	313	386:46	0:09	66:19
1TZ8	127	53:22	1:50	34:49
1UNL	292	421:11	0:09	24:59
1X8X	322	100:26	0:12	21:43
1Y6B	366	353:19	0:06	24:56
2BR1	297	253:31	0:11	24:15
Average	-	223:57	0:19	28:33

From the Table 3.2, it is clearly seen that our new algorithms run much faster than the AutoDock, especially the truncate search algorithm. Average computational time of AutoDock, Truncate search algorithm, and Full search algorithm is 223:57, 0:19, and 28:33 minutes, respectively, which correspond to a speed up ratio of 690 and 8 for truncate and full search algorithms. Although the computer used for AutoDock and our search algorithms are somewhat slightly different in specifications, i.e. CPU, RAM, operating system, this average computational time can give a clue about performance of our algorithms. Moreover, our algorithms are feasible for parallel computation, which will dramatically decrease the calculation time; while the AutoDock is not capable for parallel running.

Apart from the speed (calculation time), accuracy of the algorithm is another important factor. The accuracy of docking algorithm is generally evaluated by comparing the docked configuration to the corresponding X-ray structure. Therefore, the docked configurations predicted from our two search algorithms for all 15 compounds in the test set were compared with the X-ray structures. Results are displayed in Figure 3.1 to 3.15.

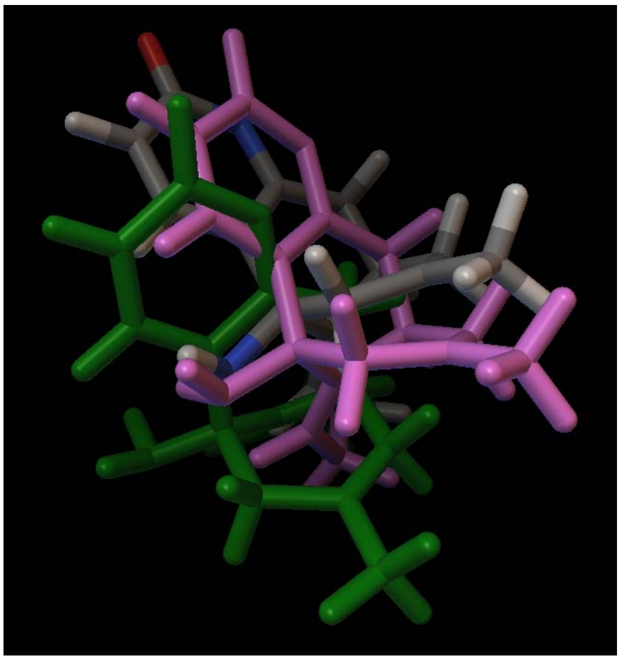


Figure 3.1 Comparison of ligand configurations in 1GPK between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

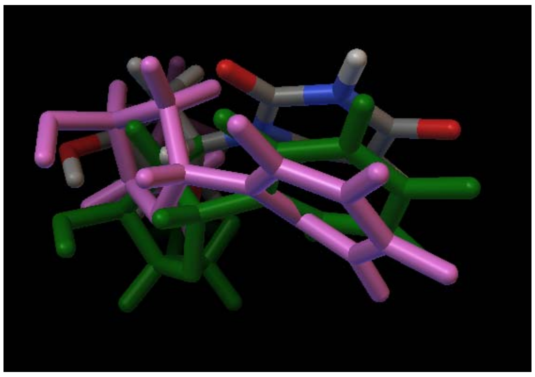


Figure 3.2 Comparison of ligand configurations in 1HVY between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

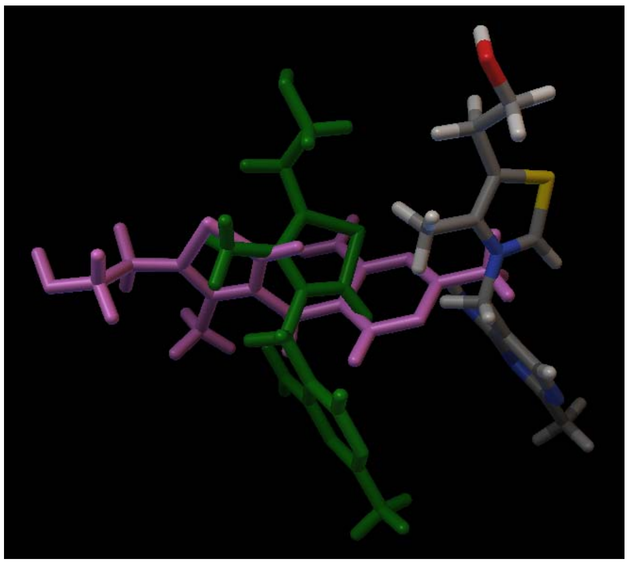


Figure 3.3 Comparison of ligand configurations in 1IG3 between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

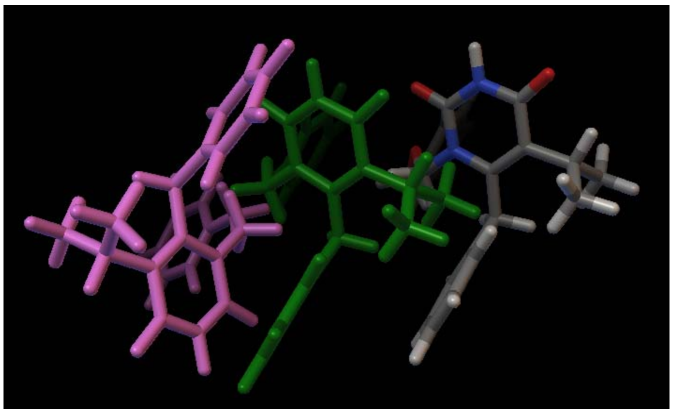


Figure 3.4 Comparison of ligand configurations in 1JLA between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

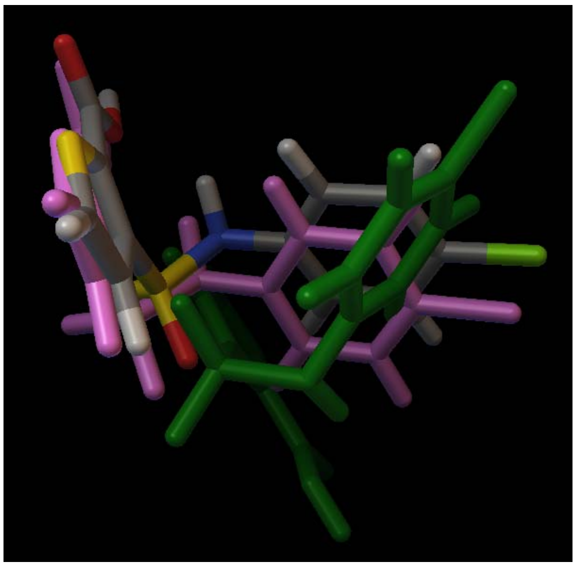


Figure 3.5 Comparison of ligand configurations in 1L2S between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

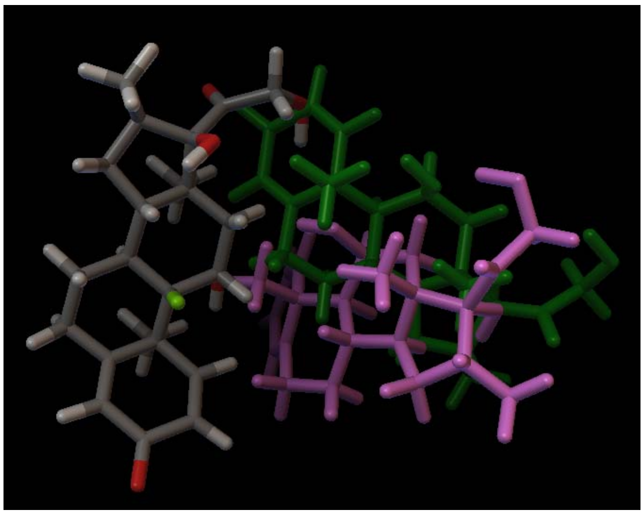


Figure 3.6 Comparison of ligand configurations in 1M2Z between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

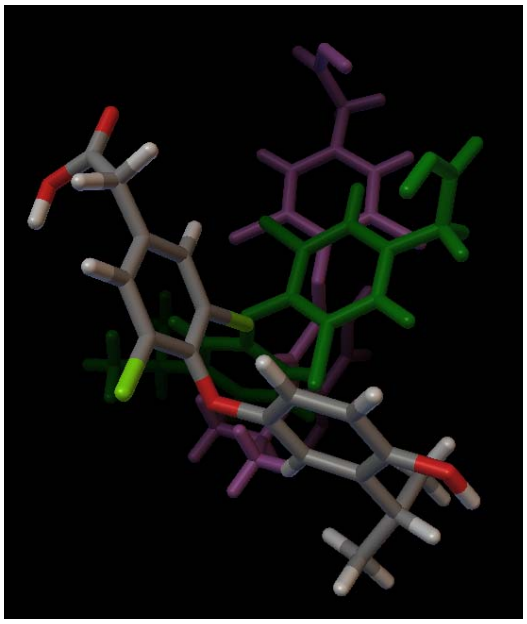


Figure 3.7 Comparison of ligand configurations in 1NAV between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

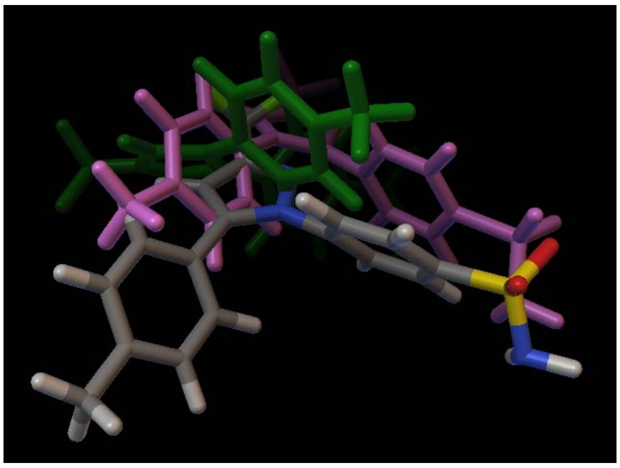


Figure 3.8 Comparison of ligand configurations in 1OQ5 between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

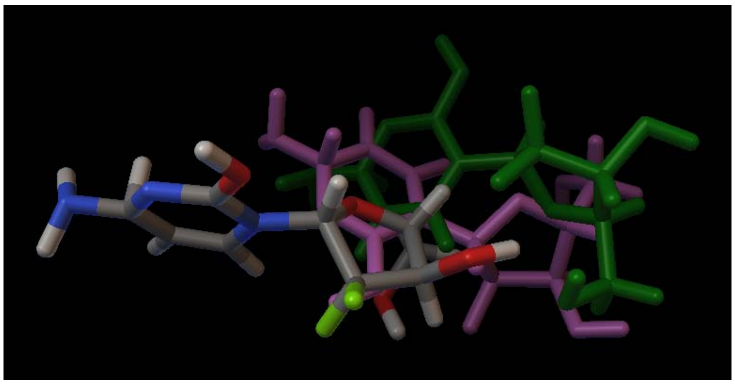


Figure 3.9 Comparison of ligand configurations in 1P62 between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

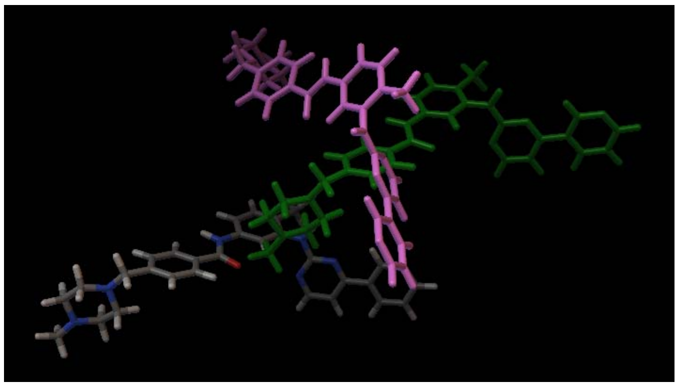


Figure 3.10 Comparison of ligand configurations in 1T46 between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

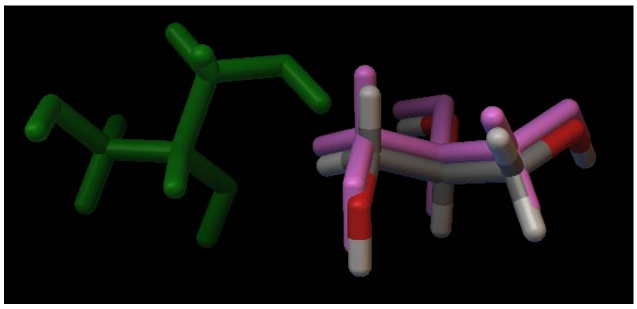


Figure 3.11 Comparison of ligand configurations in 1TZ8 between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

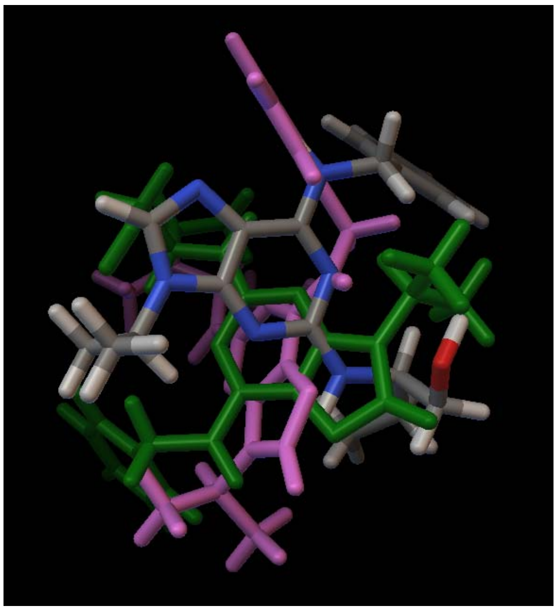


Figure 3.12 Comparison of ligand configurations in 1UNL between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

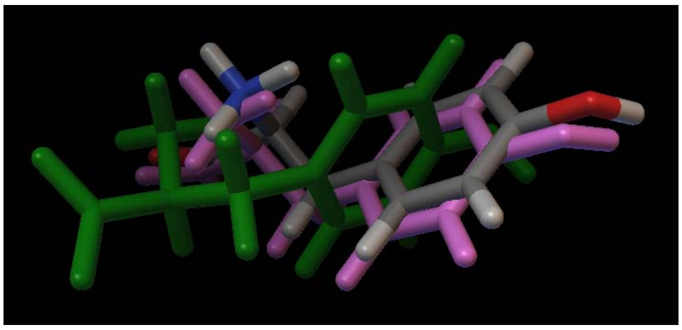


Figure 3.13 Comparison of ligand configurations in 1X8X between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

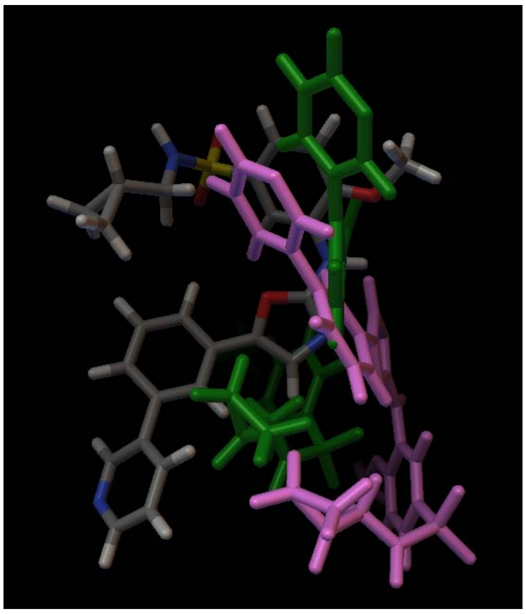


Figure 3.14 Comparison of ligand configurations in 1Y6B between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

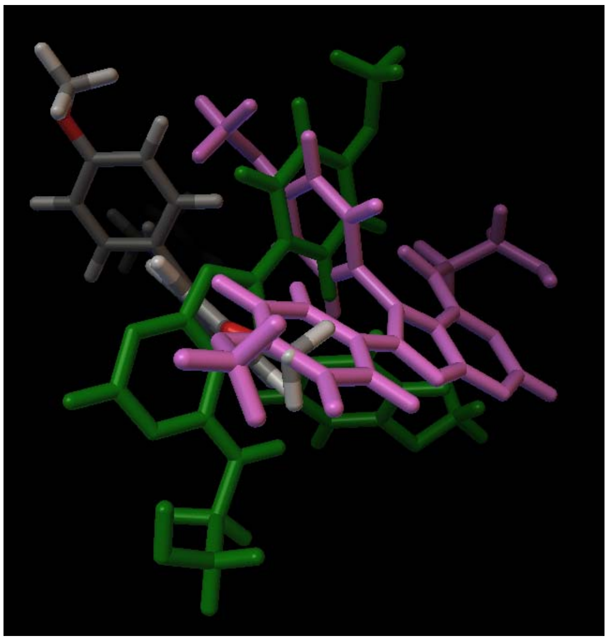


Figure 3.15 Comparison of ligand configurations in 2BR1 between X-ray structure (green color) and docked structures, using our new algorithm with full search (pink color) and truncate search (color by elements).

From all the figures, our new search algorithms can predict the conformations of ligand (pink and atom-type-based color) quite close to the experimental data (green color), at least within the same area except 1TZ8. In general, the full search algorithm is better than the truncate one, for example, 1HVY and 1Y6B systems. The two algorithms are differenct in the accepting/rejecting the generated spheres. The full search explores all search space witin the gird box, thus increase a probability to find the correct answer, compared to the truncate search algorithm. Theoritically, the performance of our search algorithm can be greatly enhanced if a rotation of the vector around its own axis is incluced during the search. And this can be done for further study.

CHAPTER IV
CONCLUSION

By using devide-and-conquer methodology together with computer graphic approach, our new search algorithm for the docking calculation can run really fast (for the truncate search algorithm) and explore all search space within the grid map (for the full search algorithm) as compared to the AutoDock. Moreover, the algorithm can solve the problem of repeatability in the software that use genetic algorithm as search method. The docked configurations of all 15 compounds in the test set obtained from our algorithm are quite close to the experimental data, indicating the high performance of our algorithm. This software can guarantee optimum and complete search of the starting van der Waals grid map space.

REFERENCES

[1] Tonkens, R. An overview of the drug development process. *Physician Executive Journal* May-June 2005: 48-52.

[2] Hansch, C., A Quantitative Approach to Biochemical Structure–Activity Relationships. *Accounts of Chemical Research* 2 (1969): 232-239.

[3] Cramer, R. D. III, Patterson, D. E., and Bunce, J. D., Comparative Molecular Field Analysis (CoMFA). 1. Effect of Shape on Biding of Steroids to Carrier Proteins. *Journal of the American Chemical Soceity* 110 (1988): 5959-5967.

[4] Klebe, G., Abraham, U., and Mietzner, T., Molecular Similarity Indices in a Comparative Analysis (CoMSIA) of Drug Molecules to Correlate and Predict Their Biological Activity. *Journal of Medicinal Chemistry* 37 (1994): 4130-4146.

[5] Lengauer, T., and Rarey, M. Computational methods for biomolecular docking. *Current Opinion in Structural Biology* 6 (1996): 402-406.

[6] Yang, S.-Y., Pharmacophore modeling and application in drug discovery: challenges and recent advances. *Drug Discovery Today* 15 (2010): 444-450.

[7] Walters, W. P., Stahl, M. T., and Murcko, M. A., Virtual screening – an overview. *Drug Discovery Today* 3 (1998): 160-178.

[8] Kitchen, D.B., Decornez, H., Furr, J.R., and Bajorath, J., Docking and scoring in virtual screening for drug discovery: methods and applications. *Nature Reviews Drug Discovery* 3 (2004): 935-949.

[9] Shoichet, B.K., McGovern, S.L., Wei, B., and Irwin, J.J., Lead discovery using molecular docking. *Current Opinion in Chemical Biology* 6 (2002): 439-446.

[10] Chen, Y., and Shoichet, B.K, Molecular docking and ligand specificity in fragment-based inhibitor discovery. *Nature Chemical Biology* 5 (2009): 358-364.

[11] Meng, X.-Y., Zhang, H.-X., Mezei, M., and Cui, M., Molecular Docking: A powerful approach for structure-based drug discovery. *Current Computer-Aided Drug Design* 7 (2011): 146-157.

[12] Morris, G. M. and Lim-Wilby, M. Molecular Docking, in Andreas Kukol (ed.), Molecular Modeling of Proteins. *New Jersey: Humana Press* 2008: 365-382.

[13] Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindvalov, I.N., and Bourne, P.E., The Protein Data Bank. *Nucleic Acids Research* 28 (2000): 235-242.

[14] Halperin, I., Ma, B., Wolfson, H., and Nussinov, R., Principles of docking: An overview of search algorithms and a guide to scoring functions. *Proteins: Structure, Function, and Genetics* 47 (2002): 409-443.

[15] Taylor, R.D., Jewsbury, P.J., and Essex, J.W., A review of protein-small molecule docking methods. *Journal of Computer-Aided Molecular Design* 16 (2002): 151-166.

[16] Chen, Y., and Pohlhaus, D. T., *In silico* docking and scoring of fragments. *Drug Discovery Today: Tehnologies* 7 (2010): e149-e156.

[17] Verdonk, M. L., Giangreco, I., Hall, R. J., Korb, O., Mortenson, P. N., and Murray, C. W., Docking Performance of Fragments and Druglike Compounds. *Journal of Medicinal Chemistry* 54 (2011): 5422-5431.

[18] Li, X., Li, Y., Cheng, T., Liu, Z., and Wang, R., Evaluation of the performance of four molecular docking programs on a diverse set of protein-ligand complexes. *Journal of Computational Chemistry* 31 (2010): 2109-2125.

[19] Zhou, Z., Felts, A. K., Friesner, R. A., and Levy, R. M., Comparative Performance of Several Flexible Docking Programs and Scoring Functions: Enrichment Studies for a Diverse Set of Pharmaceutically Relevant Targets. *Journal of Chemical Information and Modeling* 47 (2007): 1599-1608.

[20] Hartshorn, M. J., Verdonk, M. L., Chessari, G., Brewerton, S. C., Mooij, W. T. M., Mortenson, P. N. and Murray, C. W., Diverse, High-Quality Test

Set for the Validation of Protein-Ligand Docking Performance. *Journal of Medicinal Chemistry* 50 (2007): 726-741.

[21] Kontoyianni, M., McClellan, L. M., and Sokol, G. S., Evaluation of Docking Performance: Comparative Data on Docking Algorithms. *Journal of Medicinal Chemistry* 47 (2004): 558-565.

[22] McConkey, B. J., Sobolev, V., and Edelman, M., The performance of current methods in ligand-protein docking. *Current Science* 83 (2002): 845-856.

[23] Norel, R., Fischer, D., Wolfson, H. J., and Nussinov, R., Molecular surface recognition by a computer vision-based technique. *Protein Engineering* 7 (1994): 39-46.

[24] Rarey, M., Kramer, B., Lengauer, T., and Klebe, G., A fast flexible docking method using an incremental construction algorithm. *Journal of Molecular Biology* 261 (1996): 470–489.

[25] Morris, G.M., Goodsell, D.S., Huey, R., and Olson, A.J., Distributed automated docking of flexible ligands to proteins: Parallel applications of AutoDock 2.4. *Journal of Computer-Aided Molecular Design* 10 (1996): 293-304.

[26] Morris, G.M., Goodsell, D.S., Halliday, R.S., Huey, R., Hart, W.E., Belew, R.K., and Olson, A.J., Automated docking using a lamarckian genetic algorithm and empirical biding free energy function. *Journal of Computational Chemistry* 19 (1998): 1632-1662.

[27] Moustakas, D.T., Lang, P.T., Pegg, S., Pettersen, E.T., Kuntz, I.D., Broojimans, N., and Rizzo, R.C., Development and validation of a modular, extensible docking program: DOCK 5. *Journal of Computer-Aided Molecular Design* 20 (2006): 601-609.

[28] Chen, R., and Weng, Z., Docking unbound proteins using shape complementarily, desolvation, and electrostatics. *Proteins: Structure, Function, and Genetics* 47 (2002): 281-294.

[29] Jones, G., Willett, P., Glen, R.C., Leach, A.R., and Taylor, R., Development and validation of a genetic algorithm for flexible docking. *Journal of Molecular Biology* 267 (1997): 727-748.

[30] Friesner, R. A., Banks, J. L., Murphy, R. B., Halgren, T. A., Klicic, J. J., Mainz, D. T., Repasky, M. P., Knoll, E. H., Shelley, M., Perry, J. K., Shaw, D. E., Francis, P., and Shenkin, P. S., Glide: A New Approach for Rapid, Accurate Docking and Scoring. 1. Method and Assessment of Docking Accuracy. *Journal of Medicinal Chemistry* 47 (2004): 1739–1749.

[31] Morris, G. M., Goodsell, D. S., Pique, M. E., Lindstrom, W. L., Huey, R., Forli, S., Hart, W. E., Halliday, S., Belew, R. K., and Olson, A. J., AutoDock Version 4.2 User Guide, (May 26, 2009).

[32] Morris, G. M., Huey, R., Lindstrom, W. L., Sanner, M. F., Belew, R. K., Goodsell, D. S., and Olson, A. J., AutoDock4 and AutoDockTools4: Automated Docking with Selective Receptor Flexibility. *Journal of Computational Chemistry* 30 (2009): 2785-2791.

[33] Huey, R., Morris, G. M., Olson, A. J., and Goodsell, D. S., A semiempirical free energy force field with charge-based desolvation. *Journal of Computational Chemistry* 28 (2007): 1145-1152.

[34] Morris, G.M., Goodsell, D.S., Huey, R., Hart, W. E., Halliday, S., Belew, R. K., and Olson, A. J., AutoDock Version 3.0.5 User Guide (November 20, 2001).

[35] Dvir, H., Jiang, H. L., Wong, D. M., Harel, M., Chetrit, M., He, X. C., Jin, G. Y., Yu, G. L., Tang, X. C., Silman, I., Bai, D. L., and Sussman, J. L., X-ray structures of Torpedo californica acetylcholinesterase complexed with (+)-huperzine A and (-)-huperzine B: structural evidence for an active site rearrangement. *Biochemistry* 41 (2002): 10810-10818.

[36] Phan, J., Koli, S., Minor, W., Dunlap, R. B., Berger, S. H., and Lebioda, L., Human thymidylate synthase is in the closed conformation when complexed with dUMP and raltitrexed, an antifolate drug. *Biochemistry* 40 (2001): 1897-1902.

[37] Timm, D. E., Liu, J., Baker, L. J., and Harris, R. A., Crystal structure of thiamin pyrophosphokinase. *Journal of Molecular Biology* 310 (2001): 195-204.

[38] Ren, J., Nichols, C., Bird, L., Chamberlain, P., Weaver, K., Short, S., Stuart, D. I., and Stammers, D. K., Structural mechanisms of drug resistance for mutations at codons 181 and 188 in HIV-1 reverse transcriptase and the improved resilience of second generation non-nucleoside inhibitors. *Journal of Molecular Biology* 312 (2001): 795-805.

[39] Powers, R. A., Morandi, F., and Shoichet, B. K., Structure-based discovery of a novel, noncovalent inhibitor of AmpC beta-lactamase. *Structure* 10 (2002): 1013-1023.

[40] Bledsoe, R. K., Montana, V. G., Stanley, T. B., Delves, C. J., Apolito, C. J., McKee, D. D., Consler, T. G., Parks, D. J., Stewart, E. L., Willson, T. M., Lambert, M. H., Moore, J. T., Pearce, K. H., and Xu, H. E., Crystal structure of the glucocorticoid receptor ligand binding domain reveals a novel mode of receptor dimerization and coactivator recognition. *Cell* 110 (2002): 93-105.

[41] Ye, L., Li, Y. L., Mellström, K., Mellin, C., Bladh, L. G., Koehler, K., Garg, N., Garcia Collazo, A. M., Litten, C., Husman, B., Persson, K., Ljunggren, J., Grover, G., Sleph, P. G., George, R., and Malm, J., Thyroid receptor ligands. 1. Agonist ligands selective for the thyroid receptor beta1. *Journal of Medicinal Chemistry* 46 (2003): 1580-1588.

[42] Weber, A., Casini, A., Heine, A., Kuhn, D., Supuran, C. T., Scozzafava, A., and Klebe, G., Unexpected nanomolar inhibition of carbonic anhydrase by COX-2-selective celecoxib: new pharmacological opportunities due to related binding site recognition. *Journal of Medicinal Chemistry* 47 (2004): 550-557.

[43] Sabini, E., Ort, S., Monnerjahn, C., Konrad, M., and Lavie, A., Structure of human dCK suggests strategies to improve anticancer and antiviral therapy. *Nature Structural Biology* 10 (2003): 513-519.

[44] Mol, C. D., Dougan, D. R., Schneider, T. R., Skene, R. J., Kraus, M. L., Scheibe, D. N., Snell, G. P., Zou, H., Sang, B. C., and Wilson, K. P., Structural basis for the autoinhibition and STI-571 inhibition of c-Kit tyrosine kinase. *Journal of Biological Chemistry* 279 (2004): 31655-31663.

[45] Morais-de-Sá, E., Pereira, P. J., Saraiva, M. J., and Damas, A. M., The crystal structure of transthyretin in complex with diethylstilbestrol: a promising template for the design of amyloid inhibitors. *Journal of Biological Chemistry* 279(2004): 53483-53490.

[46] Mapelli, M., Massimiliano, L., Crovace, C., Seeliger, M. A., Tsai, L. H., Meijer, L., and Musacchio, A., Mechanism of CDK5/p25 binding by CDK inhibitors. *Journal of Medicinal Chemistry* 48 (2005): 671-679.

[47] Kobayashi, T., Takimura, T., Sekine, R., Kelly, V. P., Kamata, K., Sakamoto, K., Nishimura, S., and Yokoyama, S., Structural snapshots of the KMSKS loop rearrangement for amino acid activation by bacterial tyrosyl-tRNA synthetase. *Journal of Biological Chemistry* 346 (2005): 105-117.

[48] Harris, P. A., Cheung, M., Hunter, R. N., Brown, M. L., Veal, J. M., Nolte, R. T., Wang, L., Liu, W., Crosby, R. M., Johnson, J. H., Epperly, A. H., Kumar, R., Luttrell, D. K., and Stafford, J. A., Discovery and evaluation of 2-anilino-5-aryloxazoles as a novel class of VEGFR2 kinase inhibitors. *Journal of Medicinal Chemistry* 48 (2005): 1610-1619.

[49] Foloppe, N., Fisher, L. M., Howes, R., Kierstan, P., Potter, A., Robertson, A. G., and Surgenor, A. E., Structure-based design of novel Chk1 inhibitors: insights into hydrogen bonding and protein-ligand affinity. *Journal of Medicinal Chemistry* 48 (2005): 4332-4345.

APPENDICIES

Appendix A

Source Code of MapMaker.java

```
public class MapMaker implements MapConstant {
    private int numberOfCoXYZ = 0;      // number of Atom in ligand
    private int endOfCo = 0;            // end of Co-ordinate
    private String inmap = "";          // String that contain Map
    private String[] elemSymbol;        // String Array for Element Symbol
    private int[] elemSymbolIs;         // Integer Array that encoding Element Symbol
    private float[] charge;             // Floating point Array that contain Charge
    private Ligand oriLgCo;             // Ligand that contain original ligand co-ordinate
    private Ligand allLgAtmAtOrgArray[];// Ligand Array that contain all Ligand
Atom at Origin ()
   private float rMaximum[];           // Parallel Array to keep r at Maximum length
    private int longestIndex[];        // Parallel Array to keep index of the longest
distance
    public LSphereData[] lsd;           // Array of LSphereData which all point different
at 1 degree.

    // Constructor for MapMaker.
    // This will set the state of oriLgCo (original Ligand Co-ordinate)
    // and set all Ligand Atom to the Co-ordinate (0, 0, 0)
    public MapMaker(String inmapFileLocation) {
            System.out.println("MapMaker Constructor");
            this.inmap = new String(this.prepareMap(inmapFileLocation));
            endOfCo = this.inmap.indexOf("END");
            this.oriLgCo = initLigand(this.inmap);
this.rMaximum = new float[this.numberOfCoXYZ];
this.longestIndex = new int[this.numberOfCoXYZ]; // keeping  index of Atom which
stays at the fastest of this index.
this.lsd = new LSphereData[this.numberOfCoXYZ]; // keeping data of ligand sphere
        this.setAllLgAtmToOrigin(this.oriLgCo);
//   This loop is for finding maximum radius in ligand and initiating lsd.
 for (int i = 0; i < this.numberOfCoXYZ; i++) {
        this.findRMax(this.allLgAtmAtOrgArray[i], i);
           this.lsd[i] = new LSphereData(i, longestIndex[i], rMaximum[i]);
        }
        System.out.println("End MapMaker Constructor");
    }

    //Show inmap
    public void showInMap() {
        System.out.println("This is and inmap.\n" + this.inmap);
    }

    // get Element Symbol
    public String[] getElemSymbol() {
        return this.elemSymbol;
    }

    // get Original Ligand Co-ordinate
    public Ligand getOriLg() {
        return this.oriLgCo;
    }

    // This method will instantiate and return "Original Ligand.pdbqt" file.
    public String getLigandPDBQT() {
        String tInmap = new String(this.inmap);
        return tInmap;
    }

    // get String of Element Symbol. rowI start from index 0.
    // Element Symbol is the last Column of PDB or PDBQT File
```

```
public String getElementType(int rowI) {
   return this.elemSymbol[rowI];
}

// get String Array of Element Symbol Array
public String[] getElementTypeArray() {
   return this.elemSymbol;
}

// get Integer Array of Element Symbol Array
public int[] getElemTypeIs() {
   return this.elemSymbolIs;
}

// This method will Change Element Symbol to integer form (encoding Ex. OA = 1)
private void changeESsToEIs() {
   this.elemSymbolIs = new int[this.elemSymbol.length];
   for (int i = 0; i < this.elemSymbol.length; i++) {
      if (this.elemSymbol[i].equalsIgnoreCase("OA")) {
         this.elemSymbolIs[i] = 1;
      } else if (this.elemSymbol[i].equalsIgnoreCase("C")) {
         this.elemSymbolIs[i] = 2;
      } else if (this.elemSymbol[i].equalsIgnoreCase("A")) {
         this.elemSymbolIs[i] = 3;
      } else if (this.elemSymbol[i].equalsIgnoreCase("HD")) {
         this.elemSymbolIs[i] = 4;
      } else if (this.elemSymbol[i].equalsIgnoreCase("H")) {
         this.elemSymbolIs[i] = 5;
      } else if (this.elemSymbol[i].equalsIgnoreCase("NA")) {
         this.elemSymbolIs[i] = 6;
      } else if (this.elemSymbol[i].equalsIgnoreCase("N")) {
         this.elemSymbolIs[i] = 7;
```

```
        } else if (this.elemSymbol[i].equalsIgnoreCase("SA")) {
            this.elemSymbolIs[i] = 8;
        } else if (this.elemSymbol[i].equalsIgnoreCase("S")) {
            this.elemSymbolIs[i] = 9;
        }
    }
}

// This method prepares Map from input Map location in to String type
// then attachs with String "END" at the end of file
public String prepareMap(String mapLocation) {
    String tempMap;
    String inputPDBQT = CUDockUtil.loadFile(mapLocation);
    int indexH = 0;
    int indexA = 0;
    int indexEOLA = 0;
    int indexEOLH = 0;
    StringBuffer sbMapLine = new StringBuffer("");
    indexH = inputPDBQT.indexOf("HETATM", indexH);
    indexA = inputPDBQT.indexOf("ATOM", indexA);
    indexEOLA = inputPDBQT.indexOf("\n", indexA);
    indexEOLH = inputPDBQT.indexOf("\n", indexH);
    while (indexA != -1 || indexH != -1) {
        if ((indexA < indexH || indexH == -1) && indexA != -1) {
            sbMapLine.append(inputPDBQT, indexA, indexEOLA + 1);
            indexA += 4;
            indexA = inputPDBQT.indexOf("ATOM", indexA);
            indexEOLA = inputPDBQT.indexOf("\n", indexA);
        } else if ((indexH < indexA || indexA == -1) && indexH != -1) {
            sbMapLine.append(inputPDBQT, indexH, indexEOLH + 1);
            indexH += 6;
            indexH = inputPDBQT.indexOf("HETATM", indexH);
            indexEOLH = inputPDBQT.indexOf("\n", indexH);
          }
        this.numberOfCoXYZ++;

      }
      sbMapLine.append("END");
      tempMap = sbMapLine.toString();
      return tempMap;
   }

   // get the number of atoms in Ligand
   public int getNumOfAtomInLigand() {
      return this.numberOfCoXYZ;
   }

   // get Charge Array
   public float[] getChargeArray() {
      return this.charge;
   }

   // Initialize Ligand from Map String
   // output: ligand (keep coordinate x, y, z in array), elemSymbol[],
elementSymbolIs, charge[],
   public Ligand initLigand(String map) {
      StringBuffer x = new StringBuffer("01234567");
      StringBuffer y = new StringBuffer("yyyyyyyy");
      StringBuffer z = new StringBuffer("zzzzzzzz");
      StringBuffer c = new StringBuffer("01234567");

      float[] xCoArray = new float[numberOfCoXYZ];
      float[] yCoArray = new float[numberOfCoXYZ];
      float[] zCoArray = new float[numberOfCoXYZ];
```

```
        this.elemSymbol = new String[this.numberOfCoXYZ];
        this.charge = new float[numberOfCoXYZ];

        int indexH = 0;
        int indexA = 0;
        int indexEOLA = 0;
        int indexEOLH = 0;
        StringBuffer sbMapLine = new StringBuffer("");
        indexH = map.indexOf("HETATM", indexH);
        indexA = map.indexOf("ATOM", indexA);
        indexEOLA = map.indexOf("\n", indexA);
        indexEOLH = map.indexOf("\n", indexH);
        int j = 0;
        while (indexA != -1 || indexH != -1) {
            if ((indexA < indexH || indexH == -1) && indexA != -1) {
                xCoArray[j] = Float.parseFloat(x.replace(0, 8, map.substring(indexA +
pdbXCoS, indexA + pdbXCoE)).toString());
                yCoArray[j] = Float.parseFloat(y.replace(0, 8, map.substring(indexA +
pdbYCoS, indexA + pdbYCoE)).toString());
                zCoArray[j] = Float.parseFloat(z.replace(0, 8, map.substring(indexA +
pdbZCoS, indexA + pdbZCoE)).toString());
                charge[j] = Float.parseFloat(c.replace(0, 8, map.substring(indexA +
chargeS, indexA + chargeE)).toString());
                this.elemSymbol[j++] = map.substring(indexA + elemSymS, indexA +
elemSymE).trim(); //77 , 79
                indexA += 4;
                indexA = map.indexOf("ATOM", indexA);
                indexEOLA = map.indexOf("\n", indexA);
            } else if ((indexH < indexA || indexA == -1) && indexH != -1) {
                xCoArray[j] = Float.parseFloat(x.replace(0, 8, map.substring(indexH +
pdbXCoS, indexH + pdbXCoE)).toString());
```

```
                yCoArray[j] = Float.parseFloat(y.replace(0, 8, map.substring(indexH +
pdbYCoS, indexH + pdbYCoE)).toString());
                zCoArray[j] = Float.parseFloat(z.replace(0, 8, map.substring(indexH +
pdbZCoS, indexH + pdbZCoE)).toString());
                charge[j] = Float.parseFloat(c.replace(0, 8, map.substring(indexH +
chargeS, indexH + chargeE)).toString()); //7
                this.elemSymbol[j++] = map.substring(indexH + elemSymS, indexH +
elemSymE).trim();
                indexH += 6;
                indexH = map.indexOf("HETATM", indexH);
                indexEOLH = map.indexOf("\n", indexH);
            }
        }
        this.changeESsToEIs();
        Ligand lg = new Ligand(xCoArray, yCoArray, zCoArray);
        return lg;
    }

    //This method for generating PDBQT Map file
    public void genPDBQTMap(Ligand lg, String outFile)// throws IOException
    {
        try {
            PrintStream out = new PrintStream(new File(outFile));
            StringBuffer aBuffer = new StringBuffer();
            aBuffer.append(this.inmap);
            int pxs = pdbXCoS - 1;
            int pys = pdbYCoS - 1;
            int pzs = pdbZCoS - 1;
            int pxe = pdbXCoE;
            int pye = pdbYCoE;
            int pze = pdbZCoE;
            int eolIndex = 0;
            int neolIndex = 0;

            for (int i = 0; i < lg.numAtom(); i++) {
                aBuffer.replace(pxs, pxe, CUDockUtil.eightDigit(lg.coAtom('x', i)));
                aBuffer.replace(pys, pye, CUDockUtil.eightDigit(lg.coAtom('y', i)));
                aBuffer.replace(pzs, pze, CUDockUtil.eightDigit(lg.coAtom('z', i)));
                neolIndex = aBuffer.indexOf("\n", eolIndex);

                if (neolIndex - eolIndex == 78) {
                    pxs += nextCo;
                    pys += nextCo;
                    pzs += nextCo;
                    pxe += nextCo;
                    pye += nextCo;
                    pze += nextCo;
                    eolIndex = neolIndex + 1;
                } else if (neolIndex - eolIndex == 79) {
                    pxs += nextCo + 1;
                    pys += nextCo + 1;
                    pzs += nextCo + 1;
                    pxe += nextCo + 1;
                    pye += nextCo + 1;
                    pze += nextCo + 1;
                    eolIndex = neolIndex + 1;
                }
            }
            out.println(aBuffer.toString());
            out.close();
        } catch (IOException e) {
            System.out.println("Error opening output file ");
            System.exit(0);
        }
```

```
}

//This method for generating PDB Map file
public void genPDBMap(Ligand lg, String outFile)// throws IOException
{
   try {
      PrintStream out = new PrintStream(new File(outFile));
      StringBuffer aBuffer = new StringBuffer();
      aBuffer.append(this.inmap);
      int pxs = pdbXCoS - 1;
      int pys = pdbYCoS - 1;
      int pzs = pdbZCoS - 1;
      int pxe = pdbXCoE;
      int pye = pdbYCoE;
      int pze = pdbZCoE;

      for (int i = 0; i < lg.numAtom(); i++) {
         aBuffer.replace(pxs, pxe, CUDockUtil.eightDigit(lg.coAtom('x', i)));
         aBuffer.replace(pys, pye, CUDockUtil.eightDigit(lg.coAtom('y', i)));
         aBuffer.replace(pzs, pze, CUDockUtil.eightDigit(lg.coAtom('z', i)));
         pxs += nextCo;
         pys += nextCo;
         pzs += nextCo;
         pxe += nextCo;
         pye += nextCo;
         pze += nextCo;
      }
      out.println(aBuffer.toString());
      out.close();
   } catch (IOException e) {
      System.out.println("Error opening output file ");
      System.exit(0);
```

```
    }
}

// This method will set all Atom in Ligand to the Co-ordinate (0, 0, 0)
public void setAllLgAtmToOrigin(Ligand lg) {
    this.allLgAtmAtOrgArray = new Ligand[lg.numAtom()];
    Ligand tempLg;
    for (int i = 0; i < lg.numAtom(); i++) {
        tempLg = this.setToOrigin(lg, i);
        this.allLgAtmAtOrgArray[i] = tempLg;
    }
}

// This getter method returns Ligand which input index Atom of this method
// will translat to the Co-ordinate (0, 0, 0). Index start from 0.
public Ligand getEachLgAtmAtOrigin(int i) {
    return this.allLgAtmAtOrgArray[i];
}

// This Setter Method receives "Ligand, and the order of Atom in Ligand"
// and translates to the Co-ordinate (0, 0, 0)
public Ligand setToOrigin(Ligand lg, int orderOfAtom) {
    Ligand atOrigin;
    float newXCo[] = new float[lg.numAtom()];
    float newYCo[] = new float[lg.numAtom()];
    float newZCo[] = new float[lg.numAtom()];

    float xToOrigin = lg.coAtom('x', orderOfAtom);
    float yToOrigin = lg.coAtom('y', orderOfAtom);
    float zToOrigin = lg.coAtom('z', orderOfAtom);

    for (int i = 0; i < lg.numAtom(); i++) {
            newXCo[i] = lg.coAtom('x', i) - xToOrigin;
            newYCo[i] = lg.coAtom('y', i) - yToOrigin;
            newZCo[i] = lg.coAtom('z', i) - zToOrigin;
        }

        atOrigin = new Ligand(newXCo, newYCo, newZCo);
        return atOrigin;
    }

    // This method uses for finding Maximum radius of ligand atom from original co-
ordinate.
    // It will initialize rMaximum[] and longestIndex[] as well.
    public void findRMax(Ligand lOrigin, int atIndex) {
        float rMax = 0.f;
        float rTemp = 0.f;
        int farthestIndex = 0;

        for (int i = 0; i < this.getNumOfAtomInLigand(); i++) {
            rTemp = (float) Math.sqrt(Math.pow((lOrigin.coAtom('x', atIndex) -
lOrigin.coAtom('x', i)), 2)
                    + Math.pow((lOrigin.coAtom('y', atIndex) - lOrigin.coAtom('y', i)), 2)
                    + Math.pow((lOrigin.coAtom('z', atIndex) - lOrigin.coAtom('z', i)), 2));
            if (rTemp >= rMax) {
                farthestIndex = i;
                rMax = rTemp;
            }
        }
        this.rMaximum[atIndex] = rMax;
        this.longestIndex[atIndex] = farthestIndex;
    }
}
```

Appendix B

Source Code of LSphereData.java

```
public class LSphereData implements MapConstant {

    private float[] xA; // array of coordinated ligand about x axial.
    private float[] yA; // array of coordinated ligand about y axial.
    private float[] zA; // array of coordinated ligand about z axial.
    private int originAtomIndex = 0; // Index of origin atom which is the starting point.
    private int longestAtomIndex = 0;// Index of the longest atom which is the starting
point.
    private float rSphere = 0.f; //radius, distance from originAtomIndex to
longestAtomIndex.
    private float theraSphere[]; //Array contains each thera angle.
    private float phiSphere[]; //Array contains each phi angle.
    private static final int numPointOfOneDegree = 64442; //number of point in sphere
when rotate by 1 degree difference.
    private int count = 0;      // This "count index" will keep track of x,y,x, thera and phi

    // This constructor get 3 parameter which is an origin atom index,
    // a longest atom index, length of maximum radius from origin atom to longest
atom in ligand.
    public LSphereData(int originAtomI, int longestAtomI, float rS) {
        this.originAtomIndex = originAtomI;
        this.longestAtomIndex = longestAtomI;
        this.rSphere = rS;
        this.sphereCo(rS);

    }

    // This method generates xA,yA,zA Coordinate on Sphere surface with radius r
    // It will return all array of theraSphere and phiSphere as well.
    public void sphereCo(float r)//, double thera, double phi)
    {

        this.xA = new float[numPointOfOneDegree];
        this.yA = new float[numPointOfOneDegree];
        this.zA = new float[numPointOfOneDegree];
        this.theraSphere = new float[this.numPointOfOneDegree];
        this.phiSphere = new float[this.numPointOfOneDegree];

        xA[count] = (float) (r * Math.sin(Math.toRadians(0)) *
Math.cos(Math.toRadians(0)));
        yA[count] = (float) (r * Math.sin(Math.toRadians(0)) *
Math.sin(Math.toRadians(0)));
        zA[count] = (float) (r * Math.cos(Math.toRadians(0)));
        this.theraSphere[count] = 0.f;
        this.phiSphere[count] = 0.f;
        count++;

        for (int thera = 1; thera < 180; thera++) {
            for (int phi = 0; phi < 360; phi++) {
                if (thera == 90 && phi == 90) {
                    xA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * cos90);
                    yA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * sin90);
                    zA[count] = (float) (r * cos90);
                    this.theraSphere[count] = 90;
                    this.phiSphere[count] = 90;
                    count++;
                } else if (thera == 90 && phi == 180) {
                    xA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * cos180);
                    yA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * sin180);
                    zA[count] = (float) (r * cos90);
                    this.theraSphere[count] = 90;
                    this.phiSphere[count] = 180;
                    count++;
                } else if (thera == 90 && phi == 270) {
                    xA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * cos270);
                    yA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * sin270);
                    zA[count] = (float) (r * cos90);
                    this.theraSphere[count] = 90;
                    this.phiSphere[count] = 270;
                    count++;
                } else if (thera == 90) {
                    xA[count] = (float) (r * Math.sin(Math.toRadians(thera)) *
Math.cos(Math.toRadians(phi)));
                    yA[count] = (float) (r * Math.sin(Math.toRadians(thera)) *
Math.sin(Math.toRadians(phi)));
                    zA[count] = (float) (r * cos90);
                    this.theraSphere[count] = 90;
                    this.phiSphere[count] = phi;
                    count++;
                } else if (phi == 90) {
                    xA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * cos90);
                    yA[count] = (float) (r * Math.sin(Math.toRadians(thera)) *
Math.sin(Math.toRadians(phi)));
                    zA[count] = (float) (r * Math.cos(Math.toRadians(thera)));
                    this.theraSphere[count] = thera;
                    this.phiSphere[count] = 90;
                    count++;
                } else if (phi == 180) {
                    xA[count] = (float) (r * Math.sin(Math.toRadians(thera)) *
Math.cos(Math.toRadians(phi)));
                    yA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * sin180);
                    zA[count] = (float) (r * Math.cos(Math.toRadians(thera)));
                    this.theraSphere[count] = thera;
                    this.phiSphere[count] = 180;
                    count++;
                } else if (phi == 270) {
                    xA[count] = (float) (r * Math.sin(Math.toRadians(thera)) * cos270);
                    yA[count] = (float) (r * Math.sin(Math.toRadians(thera)) *
Math.sin(Math.toRadians(phi)));
                    zA[count] = (float) (r * Math.cos(Math.toRadians(thera)));
                    this.theraSphere[count] = thera;
                    this.phiSphere[count] = 270;
                    count++;
                } else {
                    xA[count] = (float) (r * Math.sin(Math.toRadians(thera)) *
Math.cos(Math.toRadians(phi)));
                    yA[count] = (float) (r * Math.sin(Math.toRadians(thera)) *
Math.sin(Math.toRadians(phi)));
                    zA[count] = (float) (r * Math.cos(Math.toRadians(thera)));
                    this.theraSphere[count] = thera;
                    this.phiSphere[count] = phi;
                    count++;
                }
            }
        }

        xA[count] = (float) (r * Math.sin(Math.toRadians(180)) *
Math.cos(Math.toRadians(0)));
        yA[count] = (float) (r * Math.sin(Math.toRadians(180)) *
Math.sin(Math.toRadians(0)));
        zA[count] = (float) (r * Math.cos(Math.toRadians(180)));
        this.theraSphere[count] = 180;
        this.phiSphere[count] = 0;
      }
```

```
// This methode returns xA at the index i.
public float getX(int i) {
   return xA[i];
}

// This methode returns yA at the index i.
public float getY(int i) {
   return yA[i];
}

// This methode returns zA at the index i.
public float getZ(int i) {
   return zA[i];
}

// This methode returns originAtomIndex,
// which is the index of origin atom.
public int getOriginAtomIndex() {
   return originAtomIndex;
}

// This method returns longestAtomIndex,
// which is the index of longest atom from originAtomIndex.
public int getLongestAtomIndex() {
   return longestAtomIndex;
}

//This method returns rSphere.
public float getrSphere() {
   return rSphere;
}
```

```
//This method returns theraSphere at index i.
public float getTheraSphere(int i) {
   return theraSphere[i];
}

//This method returns phiSphere at index i.
public float getPhiSphere(int i) {
   return phiSphere[i];
}

//This method sets x to the private data xA at index i.
public void setX(float x, int i) {
   this.xA[i] = x;
}

//This method sets y to the private data yA at index i.
public void setY(float y, int i) {
   this.yA[i] = y;
}

//This method sets z to the private data zA at index i.
public void setZ(float z, int i) {
   this.zA[i] = z;
}

//This method sets originAtomIndex to the private data originAtomIndex.
public void setOriginAtomIndex(int originAtomIndex) {
   this.originAtomIndex = originAtomIndex;
}

//This method sets longestAtomIndex to the private data longestAtomIndex.
```

```
public void setLongestAtomIndex(int longestAtomIndex) {
    this.longestAtomIndex = longestAtomIndex;
}

//This method sets rSphere to the private data rSphere.
public void setrSphere(float rSphere) {
    this.rSphere = rSphere;
}

//This method sets theraSphere to the private data theraSphere array at index i.
public void setTheraSphere(float theraSphere, int i) {
    this.theraSphere[i] = theraSphere;
}

//This method sets phiSphere to the private phiSphere array at index i.
public void setPhiSphere(float phiSphere, int i) {
    this.phiSphere[i] = phiSphere;
}

//This method generates the ligand orientation, the final answer,
//from ligand input and countIn, which is an encrypted angle position.
//Algorithm is to keep the longest co-ordinate ligand atom from input lg,
//and decrypt countIn in to thera and phi angle.
//Finally the ligand will generate final answer according to thera and phi angle.
public Ligand generateLgOrient(Ligand lg, int countIn) {

    int numLgAtom = lg.numAtom();
    int thera = (int) this.getTheraSphere(countIn);
    int phi = (int) this.getPhiSphere(countIn);

    float Cx = lg.coAtom('x', this.longestAtomIndex);
    float Cy = lg.coAtom('y', this.longestAtomIndex);
```

```
        float Cz = lg.coAtom('z', this.longestAtomIndex);
        float d = (float) Math.sqrt((double) Cy * Cy + (double) Cz * Cz);

// Making the longest vector of ligand align with z axials by rotating about x axial,
// then rotating again over y axial.

//rotateOverX
        Ligand roxLg = null;
        float newX0Co[] = new float[numLgAtom];
        float newY0Co[] = new float[numLgAtom];
        float newZ0Co[] = new float[numLgAtom];
        for (int i = 0; i < numLgAtom; i++) {
            newY0Co[i] = lg.coAtom('y', i) * (lg.coAtom('z', this.longestAtomIndex) / d) -
lg.coAtom('z', i) * (Cy / d);
            newZ0Co[i] = lg.coAtom('y', i) * (Cy / d) + lg.coAtom('z', i) * (lg.coAtom('z',
this.longestAtomIndex) / d);
            newX0Co[i] = lg.coAtom('x', i);
        }
        roxLg = new Ligand(newX0Co, newY0Co, newZ0Co);
//rotateOverY
        Ligand royLg = null;
        float newXCo[] = new float[numLgAtom];
        float newYCo[] = new float[numLgAtom];
        float newZCo[] = new float[numLgAtom];
        for (int i = 0; i < numLgAtom; i++) {
            newXCo[i] = roxLg.coAtom('z', i) * roxLg.coAtom('x',
this.longestAtomIndex) / this.rSphere + lg.coAtom('x', i) * roxLg.coAtom('z',
this.longestAtomIndex) / this.rSphere;
            newYCo[i] = roxLg.coAtom('y', i); //
            newZCo[i] = roxLg.coAtom('z', i) * roxLg.coAtom('z',
this.longestAtomIndex) / this.rSphere - lg.coAtom('x', i) * roxLg.coAtom('x',
this.longestAtomIndex) / this.rSphere;
```

```
        }
        royLg = new Ligand(newXCo, newYCo, newZCo);

// Finally, adjust the angle by rotating about x axial, thera degree,
// and then rotating about y axial, phi degree.
//rotateX thera
        Ligand roxLg1 = null;
        float newX1Co[] = new float[numLgAtom];
        float newY1Co[] = new float[numLgAtom];
        float newZ1Co[] = new float[numLgAtom];
        for (int i = 0; i < numLgAtom; i++) {
            newY1Co[i] = royLg.coAtom('y', i) * (float)
(Math.cos(Math.toRadians(thera))) - royLg.coAtom('z', i) * (float)
(Math.sin(Math.toRadians(thera)));
            newZ1Co[i] = royLg.coAtom('y', i) * (float)
(Math.sin(Math.toRadians(thera))) + royLg.coAtom('z', i) * (float)
(Math.cos(Math.toRadians(thera)));
            newX1Co[i] = royLg.coAtom('x', i);
        }
        roxLg1 = new Ligand(newX1Co, newY1Co, newZ1Co);

//rotateY phi
        Ligand royLg2 = null;
        float newX2Co[] = new float[numLgAtom];
        float newY2Co[] = new float[numLgAtom];
        float newZ2Co[] = new float[numLgAtom];
        for (int i = 0; i < numLgAtom; i++) {
            newX2Co[i] = roxLg1.coAtom('z', i) * (float)
(Math.sin(Math.toRadians(phi))) + roxLg1.coAtom('x', i) * (float)
(Math.cos(Math.toRadians(phi)));
            newY2Co[i] = roxLg1.coAtom('y', i); //
            newZ2Co[i] = roxLg1.coAtom('z', i) * (float)
(Math.cos(Math.toRadians(phi))) - roxLg1.coAtom('x', i) * (float)
(Math.sin(Math.toRadians(phi)));
        }
        royLg2 = new Ligand(newX2Co, newY2Co, newZ2Co);
        return royLg2;
    }
}
```

Appendix C

Source Code of RetriveEnergy.java

```
public class RetriveEnergy {

    static Runtime r = Runtime.getRuntime();
    static long tr = r.freeMemory();
    private String gpfInfo = "";
    private String lgMapArray[];
    private String mapFileArray[];
    private float spacing;
    private String receptorFile = "";
    private String receptorName = "";
    private int nptXI = 0;
    private int nptYI = 0;
    private int nptZI = 0;
    private float gridCenterX;
    public float realGXMinD;
    private float gridCenterY;
    public float realGYMinD;
    private float gridCenterZ;
    public float realGZMinD;
    private int numGridPoint;
    /**
     * ** assigned receptor types: oxygen->OA, carbon->C, arom_carbon->A,
     * hydrogen->HD, nonHB_hydrogen->H, nitrogen->NA, nonHB_nitrogen->N,
     * sulphur->SA, nonHB_sulphur->S"
     */
    private float OAMapArray[];
    private float OAMapArray3D[][][];
    private EnCo OAMapArrayEnCo[];
    private float AMapArray[];
    private float AMapArray3D[][][];
    private EnCo AMapArrayEnCo[];
    private float CMapArray[];
    private float CMapArray3D[][][];
    private EnCo CMapArrayEnCo[];
    private float HDMapArray[];
    private float HDMapArray3D[][][];
    private EnCo HDMapArrayEnCo[];
    private float HMapArray[];
    private float HMapArray3D[][][];
    private EnCo HMapArrayEnCo[];
    private float NAMapArray[];
    private float NAMapArray3D[][][];
    private EnCo NAMapArrayEnCo[];
    private float NMapArray[];
    private float NMapArray3D[][][];
    private EnCo NMapArrayEnCo[];
    private float SAMapArray[];
    private float SAMapArray3D[][][];
    private EnCo SAMapArrayEnCo[];
    private float SMapArray[];
    private float SMapArray3D[][][];
    private EnCo SMapArrayEnCo[];
    private float eMapArray[];
    private float eMapArray3D[][][];
    private EnCo eMapArrayEnCo[];
    private float dMapArray[];
    private float dMapArray3D[][][];
    private EnCo dMapArrayEnCo[];
    private boolean lInGrid = false;
    private boolean sedGrid3DB[][][];
```

```
    private int inGrid = 0;
    private long outGrid = 0;
    private long lgInGrid = 0;

    private void constructFromString(String gpfLocate) {
        System.out.println("Start RetriveEnergy Constructor.");
        int iR4;
        String tR4Locate;
        iR4 = gpfLocate.indexOf("DockFile"); //All recepters, .pdbqt, .gpf should be in
this directory. i stand for index.
        int rcNameLength = 0;
        rcNameLength = gpfLocate.indexOf(".") - (iR4 + 9);
        tR4Locate = gpfLocate.substring(0, iR4 + 9); // 9 means 9 charactor of
("DockFile/").
        String rcName = gpfLocate.substring(iR4 + 9, iR4 + 9 + rcNameLength);
//rcName is Recepter Name
        int iLT, iSp, iGC, iR; //index of "ligand_types"+13, "spacing"+8,
"gridcenter"+11, "receptor ".
        int eLT, eSp, eGC, eR, eN;
        String lgAtomType = "";
        this.gpfInfo = CUDockUtil.loadFile(gpfLocate);
        this.gpfInfo.trim();
        iGC = this.gpfInfo.indexOf("gridcenter") + 11; // 11 is the length of "gridcenter "
        eGC = this.gpfInfo.indexOf("#", iGC);
        String tGC[] = this.gpfInfo.substring(iGC, eGC).split(" ");
        this.gridCenterX = Float.parseFloat(tGC[0]);
        this.gridCenterY = Float.parseFloat(tGC[1]);
        this.gridCenterZ = Float.parseFloat(tGC[2]);
        eN = this.gpfInfo.indexOf("#", 5); // 5 is the length of "npts "
        String tnpts[] = this.gpfInfo.substring(5, eN).split(" ");
        this.nptXI = Integer.parseInt(tnpts[0]) + 1;
        this.nptYI = Integer.parseInt(tnpts[1]) + 1;
        this.nptZI = Integer.parseInt(tnpts[2]) - 1;
        this.numGridPoint = this.nptXI * this.nptYI * this.nptZI;
        iLT = this.gpfInfo.indexOf("ligand_types") + 13;        // 13 is the length of
"ligand_types "
        eLT = this.gpfInfo.indexOf("#", iLT);
        lgAtomType = this.gpfInfo.substring(iLT, eLT).trim();
        iSp = this.gpfInfo.indexOf("spacing") + 8; // 8 is the length of "spacing "
        eSp = this.gpfInfo.indexOf("#", iSp);
        this.spacing = Float.parseFloat(this.gpfInfo.substring(iSp, eSp).trim());
        this.lgMapArray = lgAtomType.split(" ");
        iR = this.gpfInfo.indexOf("receptor ", eLT) + 9;
        eR = eLT = this.gpfInfo.indexOf("#", iR);
        this.receptorFile = this.gpfInfo.substring(iR, eR).trim();
        this.receptorName = this.receptorFile.substring(0,
this.receptorFile.indexOf(".pdb"));
        this.mapFileArray = new String[this.lgMapArray.length + 2]; // 2 คือ e map กับ d
map
        for (int i = 0; i < this.lgMapArray.length; i++) {
          this.mapFileArray[i] = this.receptorName + "." + this.lgMapArray[i] + ".map";
        }
        this.mapFileArray[this.lgMapArray.length] = this.receptorName + ".e.map";
        this.mapFileArray[this.lgMapArray.length + 1] = this.receptorName + ".d.map";

        String tR4LocateA[] = new String[this.mapFileArray.length];
        for (int i = 0; i < this.mapFileArray.length; i++) {
          tR4LocateA[i] = tR4Locate + this.mapFileArray[i];
          if (tR4LocateA[i].contains(".C.map")) {
            System.out.println(tR4LocateA[i]);
            this.CMapArray = this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
            this.CMapArray3D = this.oneDTo3DMatrix(CMapArray, nptZI, nptYI,
nptXI);
```

```
            this.CMapArrayEnCo = this.makeEnCos(this.CMapArray3D, nptZI, nptYI,
nptXI);
        } else if (tR4LocateA[i].contains(".HD.map")) {
            System.out.println(tR4LocateA[i]);
            this.HDMapArray =
this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
            this.HDMapArray3D = this.oneDTo3DMatrix(HDMapArray, nptZI, nptYI,
nptXI);
            this.HDMapArrayEnCo = this.makeEnCos(this.HDMapArray3D, nptZI,
nptYI, nptXI);
        } else if (tR4LocateA[i].contains(".N.map")) {
            System.out.println(tR4LocateA[i]);
            this.NMapArray = this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
            this.NMapArray3D = this.oneDTo3DMatrix(NMapArray, nptZI, nptYI,
nptXI);
            this.NMapArrayEnCo = this.makeEnCos(this.NMapArray3D, nptZI, nptYI,
nptXI);
        } else if (tR4LocateA[i].contains(".A.map")) {
            System.out.println(tR4LocateA[i]);
            this.AMapArray = this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
            this.AMapArray3D = this.oneDTo3DMatrix(AMapArray, nptZI, nptYI,
nptXI);
            this.AMapArrayEnCo = this.makeEnCos(this.AMapArray3D, nptZI, nptYI,
nptXI);
        } else if (tR4LocateA[i].contains(".NA.map")) {
            System.out.println(tR4LocateA[i]);
            this.NAMapArray =
this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
            this.NAMapArray3D = this.oneDTo3DMatrix(NAMapArray, nptZI, nptYI,
nptXI);
            this.NAMapArrayEnCo = this.makeEnCos(this.NAMapArray3D, nptZI,
nptYI, nptXI);
```

```
        } else if (tR4LocateA[i].contains(".OA.map")) {
          System.out.println(tR4LocateA[i]);
          this.OAMapArray =
this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
          this.OAMapArray3D = this.oneDTo3DMatrix(OAMapArray, nptZI, nptYI,
nptXI);
          this.OAMapArrayEnCo = this.makeEnCos(this.OAMapArray3D, nptZI,
nptYI, nptXI);
        } else if (tR4LocateA[i].contains(".H.map")) {
          System.out.println(tR4LocateA[i]);
          this.HMapArray = this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
          this.HMapArray3D = this.oneDTo3DMatrix(HMapArray, nptZI, nptYI,
nptXI);
          this.HMapArrayEnCo = this.makeEnCos(this.HMapArray3D, nptZI, nptYI,
nptXI);
        } else if (tR4LocateA[i].contains(".S.map")) {
          System.out.println(tR4LocateA[i]);
          this.SMapArray = this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
          this.SMapArray3D = this.oneDTo3DMatrix(SMapArray, nptZI, nptYI,
nptXI);
          this.SMapArrayEnCo = this.makeEnCos(this.SMapArray3D, nptZI, nptYI,
nptXI);
        } else if (tR4LocateA[i].contains(".SA.map")) {
          System.out.println(tR4LocateA[i]);
          this.SAMapArray =
this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
          this.SAMapArray3D = this.oneDTo3DMatrix(SAMapArray, nptZI, nptYI,
nptXI);
          this.SAMapArrayEnCo = this.makeEnCos(this.SAMapArray3D, nptZI,
nptYI, nptXI);
        } else if (tR4LocateA[i].contains(".e.map")) {
          System.out.println(tR4LocateA[i]);
```

```
                this.eMapArray = this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
                this.eMapArray3D = this.oneDTo3DMatrix(eMapArray, nptZI, nptYI,
nptXI);
                this.eMapArrayEnCo = this.makeEnCos(this.eMapArray3D, nptZI, nptYI,
nptXI);
            } else if (tR4LocateA[i].contains(".d.map")) {
                System.out.println(tR4LocateA[i]);
                this.dMapArray = this.makeArMap(CUDockUtil.loadFile(tR4LocateA[i]));
                this.dMapArray3D = this.oneDTo3DMatrix(dMapArray, nptZI, nptYI,
nptXI);
                this.dMapArrayEnCo = this.makeEnCos(this.dMapArray3D, nptZI, nptYI,
nptXI);
            }
        }
        this.realGXMinD = this.gridCenterX - Math.abs((this.nptXI / 2) * this.spacing);
        this.realGYMinD = this.gridCenterY - Math.abs((this.nptYI / 2) * this.spacing);
        this.realGZMinD = this.gridCenterZ - Math.abs((this.nptZI / 2) * this.spacing);
        System.out.println("End RetriveEnergy Constr.");
    }

    public RetriveEnergy(String gpfLocate)
    {
        this.constructFromString(gpfLocate);
    }

    public float getRealGXMinD() {
        return realGXMinD;
    }

    public float getRealGYMinD() {
        return realGYMinD;
    }
    public float getRealGZMinD() {
        return realGZMinD;
    }

    public float getSpacingD() {
        return spacing;
    }

    //make EnCo Array, Energy Co-ordinate array, each array contain energy at that
grid point
    public EnCo[] makeEnCos(float[][][] map3D, int maxZI, int maxYI, int maxXI) {
        EnCo[] enCos = new EnCo[this.numGridPoint];
        int i = 0;
        for (int zI = 0; zI < maxZI; zI++) {
            for (int yI = 0; yI < maxYI; yI++) {
                for (int xI = 0; xI < maxXI; xI++) {
                    enCos[i] = new EnCo(map3D[zI][yI][xI], zI, yI, xI);
                    i++;
                }
            }
        }
        return enCos;
    }

    //convert one dimension matrix into 3 dimension matrix which type is float
    public float[][][] oneDTo3DMatrix(float[] linDs, int maxZI, int maxYI, int maxXI)
{
        float[][][] r3DMatrix = new float[maxZI][maxYI][maxXI];
        int i = 0;
        for (int zI = 0; zI < maxZI; zI++) {
            for (int yI = 0; yI < maxYI; yI++) {
```

```
            for (int xI = 0; xI < maxXI; xI++) {
                r3DMatrix[zI][yI][xI] = linDs[i++];
            }
        }
    }
    return r3DMatrix;
}

// make 3 dimension matric which type is boolean and set all of them to "false"
public boolean[][][] make3DBMatrix(int maxZI, int maxYI, int maxXI) {
    boolean[][][] r3DMatrix = new boolean[maxZI][maxYI][maxXI];
    int i = 0;
    for (int zI = 0; zI < maxZI; zI++) {
        for (int yI = 0; yI < maxYI; yI++) {
            for (int xI = 0; xI < maxXI; xI++) {
                r3DMatrix[zI][yI][xI] = false;
            }
        }
    }
    return r3DMatrix;
}

//make one dimension array map from input Map
public float[] makeArMap(String inMap) {
    float[] da = new float[this.numGridPoint]; //[10];
    int tcut;
    int start = 0;
    int end = 0;
    String tMap = "";
    inMap.trim();
    tcut = inMap.indexOf("CENTER");
    tcut = inMap.indexOf("\n", tcut) + 1;
    tMap = inMap.substring(tcut);
    end = tMap.indexOf("\n", start) + 1;
    for (int i = 0; i < da.length; i++) {
        end = tMap.indexOf("\n", start) + 1;
        da[i] = Float.parseFloat(tMap.substring(start, end));
        start = end;
    }
    return da;
}

//move Ligand to the input Co-ordinate x, y, z
public Ligand moveToCo(Ligand lg, float x, float y, float z) {
    Ligand atCo;
    float newXCo[] = new float[lg.numAtom()];
    float newYCo[] = new float[lg.numAtom()];
    float newZCo[] = new float[lg.numAtom()];

    for (int i = 0; i < lg.numAtom(); i++) {
        newXCo[i] = lg.coAtom('x', i) + x;
        newYCo[i] = lg.coAtom('y', i) + y;
        newZCo[i] = lg.coAtom('z', i) + z;
    }

    atCo = new Ligand(newXCo, newYCo, newZCo);
    return atCo;
}

public LEGResult[] docking2(MapMaker lgMap, int numResultI) {
    LEGResult[] legR = new LEGResult[numResultI];
    int[] atomType = lgMap.getElemTypeIs();
    float chargePDBQT[] = lgMap.getChargeArray();
    int numLgAtom = lgMap.getNumOfAtomInLigand();
```

```
        int rst = 0;    //element that was chosen to be rotated
        String mapTypeS = null;
        int numOA = 0;
        int numC = 0;
        int numA = 0;
        int numHD = 0;
        int numH = 0;
        int numNA = 0;
        int numN = 0;
        int numSA = 0;
        int numS = 0;

    // Find the minimum atom type that occure in ligand
        int minAt = numLgAtom;
        int[] t = new int[9];
        boolean[] se = new boolean[numLgAtom];
        for (int i = 0; i < lgMap.getNumOfAtomInLigand(); i++) {
            if (lgMap.getElementType(i).equalsIgnoreCase("OA")) {
                numOA++;
                t[0]++;
            } else if (lgMap.getElementType(i).equalsIgnoreCase("C")) {
                numC++;
                t[1]++;
            } else if (lgMap.getElementType(i).equalsIgnoreCase("A")) {
                numA++;
                t[2]++;
            } else if (lgMap.getElementType(i).equalsIgnoreCase("HD")) {
                numHD++;
                t[3]++;
            } else if (lgMap.getElementType(i).equalsIgnoreCase("H")) {
                numH++;
                t[4]++;
```

```
          } else if (lgMap.getElementType(i).equalsIgnoreCase("NA")) {
             numNA++;
             t[5]++;
          } else if (lgMap.getElementType(i).equalsIgnoreCase("N")) {
             numN++;
             t[6]++;
          } else if (lgMap.getElementType(i).equalsIgnoreCase("SA")) {
             numSA++;
             t[7]++;
          } else if (lgMap.getElementType(i).equalsIgnoreCase("S")) {
             numS++;
             t[8]++;
          }
       }
       Arrays.sort(t);
       for (int i = 0; i < t.length; i++) {
          if (t[i] > 0) {
             minAt = t[i];
             break;
          }
       }
       System.out.println("Redundant Atom Type occure: " + minAt + "times");
       System.out.println(numA + " " + numC + " " + numH + " " + numHD + " " +
numN + " " + numNA
             + " " + numOA + " " + numS + " " + numSA);
       if (numN > 0 && numN == minAt) {
          mapTypeS = "NMap";
          System.out.println("NMap Start");
       } else if (numNA > 0 && numNA == minAt) {
          mapTypeS = "NAMap";
          System.out.println("NAMap Start");
       } else if (numOA > 0 && numOA == minAt) {
```

```
    mapTypeS = "OAMap";
    System.out.println("OAMap Start");
} else if (numS > 0 && numS == minAt) {
    mapTypeS = "SMap";
    System.out.println("SMap Start");
} else if (numSA > 0 && numSA == minAt) {
    mapTypeS = "SAMap";
    System.out.println("SAMap Start");
} else if (numA > 0 && numA == minAt) {
    mapTypeS = "AMap";
    System.out.println("AMap Start");
} else if (numC > 0 && numC == minAt) {
    mapTypeS = "CMap";
    System.out.println("CMap Start");
} else if (numH > 0 && numH == minAt) {
    mapTypeS = "HMap";
    System.out.println("HMap Start");
} else if (numHD > 0 && numHD == minAt) {
    mapTypeS = "HDMap";
    System.out.println("HDMap Start");
}

for (int i = 0; i < numLgAtom; i++) {
    if (mapTypeS.equalsIgnoreCase(lgMap.getElementType(i) + "Map")) {
        se[i] = true;
    } else {
        se[i] = false;
    }
}


/**
```

```
 * ** assigned receptor types: oxygen->OA, carbon->C, arom_carbon->A,
 * hydrogen->HD, nonHB_hydrogen->H, nitrogen->NA, nonHB_nitrogen->N,
 * sulphur->SA, nonHB_sulphur->S"
 */
float OAMinEng;
float CMinEng;
float AMinEng;
float HDMinEng;
float HMinEng;
float NAMinEng;
float NMinEng;
float SAMinEng;
float SMinEng;
float thOA = 0.0f;
float thC = 0.0f;
float thA = 0.0f;
float thHD = 0.0f;
float thH = 0.0f;
float thNA = 0.0f;
float thN = 0.0f;
float thSA = 0.0f;
float thS = 0.0f;
EnCo[] OAMapEnCos = null;
EnCo[] CMapEnCos = null;
EnCo[] AMapEnCos = null;
EnCo[] HDMapEnCos = null;
EnCo[] HMapEnCos = null;
EnCo[] NAMapEnCos = null;
EnCo[] NMapEnCos = null;
EnCo[] SAMapEnCos = null;
EnCo[] SMapEnCos = null;
```

```
        EnCo[] startMapEnCos = new EnCo[this.numGridPoint];
        float startMapArray3D[][][];
        if (numOA > 0) {
            OAMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                OAMapEnCos[id] = new EnCo(this.OAMapArrayEnCo[id].getEnergyF(),
this.OAMapArrayEnCo[id].getZI(), this.OAMapArrayEnCo[id].getYI(),
this.OAMapArrayEnCo[id].getXI());
            }
            Arrays.sort(OAMapEnCos);
            OAMinEng = OAMapEnCos[0].getEnergyF();
            thOA = numOA * OAMinEng;
        }
        if (numC > 0) {
            CMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                CMapEnCos[id] = new EnCo(this.CMapArrayEnCo[id].getEnergyF(),
this.CMapArrayEnCo[id].getZI(), this.CMapArrayEnCo[id].getYI(),
this.CMapArrayEnCo[id].getXI());
            }
            Arrays.sort(CMapEnCos);
            CMinEng = CMapEnCos[0].getEnergyF();
            thC = numC * CMinEng;
        }
        if (numA > 0) {
            AMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                AMapEnCos[id] = new EnCo(this.AMapArrayEnCo[id].getEnergyF(),
this.AMapArrayEnCo[id].getZI(), this.AMapArrayEnCo[id].getYI(),
this.AMapArrayEnCo[id].getXI());
            }
            Arrays.sort(AMapEnCos);
            AMinEng = AMapEnCos[0].getEnergyF();
            thA = numA * AMinEng;
        }
        if (numHD > 0) {
            HDMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                HDMapEnCos[id] = new EnCo(this.HDMapArrayEnCo[id].getEnergyF(),
this.HDMapArrayEnCo[id].getZI(), this.HDMapArrayEnCo[id].getYI(),
this.HDMapArrayEnCo[id].getXI());
            }
            Arrays.sort(HDMapEnCos);
            HDMinEng = HDMapEnCos[0].getEnergyF();
            thHD = numHD * HDMinEng;
        }
        if (numH > 0) {
            HMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                HMapEnCos[id] = new EnCo(this.HMapArrayEnCo[id].getEnergyF(),
this.HMapArrayEnCo[id].getZI(), this.HMapArrayEnCo[id].getYI(),
this.HMapArrayEnCo[id].getXI());
            }
            Arrays.sort(HMapEnCos);
            HMinEng = HMapEnCos[0].getEnergyF();
            thH = numH * HMinEng;
        }
        if (numNA > 0) {
            NAMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                NAMapEnCos[id] = new EnCo(this.NAMapArrayEnCo[id].getEnergyF(),
this.NAMapArrayEnCo[id].getZI(), this.NAMapArrayEnCo[id].getYI(),
this.NAMapArrayEnCo[id].getXI());
            }
            Arrays.sort(NAMapEnCos);
            NAMinEng = NAMapEnCos[0].getEnergyF();
            thNA = numNA * NAMinEng;
        }
        if (numN > 0) {
            NMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                NMapEnCos[id] = new EnCo(this.NMapArrayEnCo[id].getEnergyF(),
this.NMapArrayEnCo[id].getZI(), this.NMapArrayEnCo[id].getYI(),
this.NMapArrayEnCo[id].getXI());
            }
            Arrays.sort(NMapEnCos);
            NMinEng = NMapEnCos[0].getEnergyF();
            thN = numN * NMinEng;
        }
        if (numSA > 0) {
            SAMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                SAMapEnCos[id] = new EnCo(this.SAMapArrayEnCo[id].getEnergyF(),
this.SAMapArrayEnCo[id].getZI(), this.SAMapArrayEnCo[id].getYI(),
this.SAMapArrayEnCo[id].getXI());
            }
            Arrays.sort(SAMapEnCos);
            SAMinEng = SAMapEnCos[0].getEnergyF();
            thSA = numSA * SAMinEng;
        }
        if (numS > 0) {
            SMapEnCos = new EnCo[this.numGridPoint];
            for (int id = 0; id < this.numGridPoint; id++) {
                SMapEnCos[id] = new EnCo(this.SMapArrayEnCo[id].getEnergyF(),
this.SMapArrayEnCo[id].getZI(), this.SMapArrayEnCo[id].getYI(),
this.SMapArrayEnCo[id].getXI());
```

```
            }
            Arrays.sort(SMapEnCos);
            SMinEng = SMapEnCos[0].getEnergyF();
            thS = numS * SMinEng;
        }
        int threshold = (int) (Math.abs(thOA + thC + thA + thHD + thH + thNA + thN +
thSA + thS));
        System.out.println("Threshold is: " + threshold);

        threshold = 0;

// Find out which one of map shoud be a start map. The start map should be the least
occurring atom type in ligand.
// Then initiate the startMapEnCos to that atom type.
        if (mapTypeS.equalsIgnoreCase("OAMap")) {
            startMapEnCos = OAMapEnCos;
            startMapArray3D = OAMapArray3D;
            rst = 1;    // rst is for telling which element should be a starting element
        } else if (mapTypeS.equalsIgnoreCase("CMap")) {
            startMapEnCos = CMapEnCos;
            startMapArray3D = CMapArray3D;
            rst = 2;
        } else if (mapTypeS.equalsIgnoreCase("AMap")) {
            startMapEnCos = AMapEnCos;
            startMapArray3D = AMapArray3D;
            rst = 3;
        } else if (mapTypeS.equalsIgnoreCase("HDMap")) {
            startMapEnCos = HDMapEnCos;
            startMapArray3D = HDMapArray3D;
            rst = 4;
        } else if (mapTypeS.equalsIgnoreCase("HMap")) {
            startMapEnCos = HMapEnCos;
            startMapArray3D = HMapArray3D;
            rst = 5;
        } else if (mapTypeS.equalsIgnoreCase("NAMap")) {
            startMapEnCos = NAMapEnCos;
            startMapArray3D = NAMapArray3D;
            rst = 6;
        } else if (mapTypeS.equalsIgnoreCase("NMap")) {
            startMapEnCos = NMapEnCos;
            startMapArray3D = NMapArray3D;
            rst = 7;
        } else if (mapTypeS.equalsIgnoreCase("SAMap")) {
            startMapEnCos = SAMapEnCos;
            startMapArray3D = SAMapArray3D;
            rst = 8;
        } else if (mapTypeS.equalsIgnoreCase("SMap")) {
            startMapEnCos = SMapEnCos;
            startMapArray3D = SMapArray3D;
            rst = 9;
        }


// Find the first index which contain energy over threshold.
        int startMapOverThMaxInd = 0;
        for (startMapOverThMaxInd = 0; startMapOverThMaxInd <
startMapEnCos.length; startMapOverThMaxInd++) {
            if (startMapEnCos[startMapOverThMaxInd].getEnergyF() > threshold) {
                System.out.println("First index over Threshold is: " +
startMapOverThMaxInd);
                break;
            }
        }
        int bufferEnCo = 6000;
        int countFirstFill = 0;
        boolean firstFill = true;
        EnCo[] tEnCo = new EnCo[bufferEnCo];
        float energy = 0.0f;
        EnCo btEnCo = null;
        int orientPos = 14;
        int tLegrSize = orientPos * 72;
        Ligand[] tempLgs1 = new Ligand[9];
        LEGResult[] tLegr = new LEGResult[tLegrSize];
        System.out.println(tLegr.length);
        int pos1 = 64442;

//------Docking Part------------
        for (int elementIndex = 0; elementIndex < numLgAtom; elementIndex++) {
            if (atomType[elementIndex] == rst)
            {
                System.out.println("Atom in Ligand at index: " + elementIndex + " " +
atomType[elementIndex]);
                for (int i = 0; i < startMapOverThMaxInd; i++) {
                    for (int spIndex = 0; spIndex < pos1; spIndex++) {
                        int migX = (int) (Math.round(lgMap.lsd[elementIndex].getX(spIndex)
/ spacing)) +
startMapEnCos[i].getXI();//(Math.round(lgMap.lsd[elementIndex].getX(spIndex)));
                        int migY = (int) (Math.round(lgMap.lsd[elementIndex].getY(spIndex)
/ spacing)) +
startMapEnCos[i].getYI();//(Math.round(lgMap.lsd[elementIndex].getY(spIndex)));
                        int migZ = (int) (Math.round(lgMap.lsd[elementIndex].getZ(spIndex) /
spacing)) +
startMapEnCos[i].getZI();//(Math.round(lgMap.lsd[elementIndex].getZ(spIndex)));
                        if (migX < 0 || migY < 0 || migZ < 0 || migX >= nptXI || migY >=
nptYI || migZ >= nptZI) {
```

```
                    this.outGrid++;
                    this.lInGrid = false;
                    break;
                }
                lInGrid = true;

                if (firstFill && (countFirstFill < bufferEnCo)) {
                    if (rst == 1) {
                        tEnCo[spIndex] = new
EnCo(OAMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                    } else if (rst == 2) {
                        tEnCo[spIndex] = new
EnCo(CMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                    } else if (rst == 3) {
                        tEnCo[spIndex] = new
EnCo(AMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                    } else if (rst == 4) {
                        tEnCo[spIndex] = new
EnCo(HDMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                    } else if (rst == 5) {
                        tEnCo[spIndex] = new
EnCo(HMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                    } else if (rst == 6) {
                        tEnCo[spIndex] = new
EnCo(NAMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                    } else if (rst == 7) {
                        tEnCo[spIndex] = new
EnCo(NMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                    } else if (rst == 8) {
                        tEnCo[spIndex] = new
EnCo(SAMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                    } else if (rst == 9) {
```

```
                    tEnCo[spIndex] = new
EnCo(SMapArray3D[migZ][migY][migX], spIndex, elementIndex);
                  }
                  countFirstFill++;
                } else if (firstFill && (countFirstFill == bufferEnCo)) {
                  firstFill = false;
                  countFirstFill++;
                  Arrays.sort(tEnCo);
                  if (rst == 1) {
                    btEnCo = new EnCo(OAMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                  } else if (rst == 2) {
                    btEnCo = new EnCo(CMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                  } else if (rst == 3) {
                    btEnCo = new EnCo(AMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                  } else if (rst == 4) {
                    btEnCo = new EnCo(HDMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                  } else if (rst == 5) {
                    btEnCo = new EnCo(HMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                  } else if (rst == 6) {
                    btEnCo = new EnCo(NAMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                  } else if (rst == 7) {
                    btEnCo = new EnCo(NMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                  } else if (rst == 8) {
                    btEnCo = new EnCo(SAMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
```

```
                } else if (rst == 9) {
                    btEnCo = new EnCo(SMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                }
                if (tEnCo[tEnCo.length - 1].getEnergyF() > btEnCo.getEnergyF()) {
                    tEnCo[tEnCo.length - 1] = btEnCo;
                    Arrays.sort(tEnCo);
                }
            }
            else if (!firstFill) {
                if (rst == 1) {
                    energy = OAMapArray3D[migZ][migY][migX];
                } else if (rst == 2) {
                    energy = CMapArray3D[migZ][migY][migX];
                } else if (rst == 3) {
                    energy = AMapArray3D[migZ][migY][migX];
                } else if (rst == 4) {
                    energy = HDMapArray3D[migZ][migY][migX];
                } else if (rst == 5) {
                    energy = HMapArray3D[migZ][migY][migX];
                } else if (rst == 6) {
                    energy = NAMapArray3D[migZ][migY][migX];
                } else if (rst == 7) {
                    energy = NMapArray3D[migZ][migY][migX];
                } else if (rst == 8) {
                    energy = SAMapArray3D[migZ][migY][migX];
                } else if (rst == 9) {
                    energy = SMapArray3D[migZ][migY][migX];
                }

                if (tEnCo[tEnCo.length - 1].getEnergyF() > energy && energy <
threshold) {
```

```
                    if (rst == 1) {
                        btEnCo = new EnCo(OAMapArray3D[migZ][migY][migX],
spIndex, elementIndex); //
                    } else if (rst == 2) {
                        btEnCo = new EnCo(CMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                    } else if (rst == 3) {
                        btEnCo = new EnCo(AMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                    } else if (rst == 4) {
                        btEnCo = new EnCo(HDMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                    } else if (rst == 5) {
                        btEnCo = new EnCo(HMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                    } else if (rst == 6) {
                        btEnCo = new EnCo(NAMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                    } else if (rst == 7) {
                        btEnCo = new EnCo(NMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                    } else if (rst == 8) {
                        btEnCo = new EnCo(SAMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                    } else if (rst == 9) {
                        btEnCo = new EnCo(SMapArray3D[migZ][migY][migX],
spIndex, elementIndex);
                    }
                    tEnCo[tEnCo.length - 1] = btEnCo;
                    Arrays.sort(tEnCo);

                }
```

```
            }
         }
      }
      Arrays.sort(tEnCo);
      System.out.println("tEnCo[0]: " + tEnCo[0]);
      System.out.println("tEnCo[lastIndex]: " + tEnCo[tEnCo.length - 1]);
      System.out.println("Buffer is: " + tEnCo.length);

      tempLgs1[0] =
lgMap.lsd[tEnCo[0].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmAtO
rigin(tEnCo[0].getElementIndex()), tEnCo[0].getSpIndex());
      tempLgs1[1] =
lgMap.lsd[tEnCo[2].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmAtO
rigin(tEnCo[2].getElementIndex()), tEnCo[2].getSpIndex());
      tempLgs1[2] =
lgMap.lsd[tEnCo[4].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmAtO
rigin(tEnCo[4].getElementIndex()), tEnCo[4].getSpIndex());
      tempLgs1[3] =
lgMap.lsd[tEnCo[8].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmAtO
rigin(tEnCo[8].getElementIndex()), tEnCo[8].getSpIndex());
      tempLgs1[4] =
lgMap.lsd[tEnCo[16].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmAt
Origin(tEnCo[16].getElementIndex()), tEnCo[16].getSpIndex());
      tempLgs1[5] =
lgMap.lsd[tEnCo[32].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmAt
Origin(tEnCo[32].getElementIndex()), tEnCo[32].getSpIndex());
      tempLgs1[6] =
lgMap.lsd[tEnCo[64].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmAt
Origin(tEnCo[64].getElementIndex()), tEnCo[64].getSpIndex());
      tempLgs1[7] =
lgMap.lsd[tEnCo[128].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmA
tOrigin(tEnCo[128].getElementIndex()), tEnCo[128].getSpIndex());
```

```
            tempLgs1[8] =
lgMap.lsd[tEnCo[256].getElementIndex()].generateLgOrient(lgMap.getEachLgAtmA
tOrigin(tEnCo[256].getElementIndex()), tEnCo[256].getSpIndex());
            lgMap.genPDBQTMap(tempLgs1[0] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[0].pdbqt");
            lgMap.genPDBQTMap(tempLgs1[1] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[1].pdbqt");
            lgMap.genPDBQTMap(tempLgs1[2] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[2].pdbqt");
            lgMap.genPDBQTMap(tempLgs1[3] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[3].pdbqt");
            lgMap.genPDBQTMap(tempLgs1[4] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[4].pdbqt");
            lgMap.genPDBQTMap(tempLgs1[5] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[5].pdbqt");
            lgMap.genPDBQTMap(tempLgs1[6] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[6].pdbqt");
            lgMap.genPDBQTMap(tempLgs1[7] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[7].pdbqt");
            lgMap.genPDBQTMap(tempLgs1[8] ,
"C:/Users/Aof/Documents/NetBeansProjects/RetriveEngV2/src/DockFile/Filter/temp
Lgs1[8].pdbqt");
        }
    }
```

```
        return legR;
    }
}
```

Appendix D

Source Code of Ligand.java

```
public class Ligand {

    private float[] xCoF; // This is an array of x coordinate of each atom in ligand.
    private float[] yCoF; // This is an array of y coordinate of each atom in ligand
    private float[] zCoF; // This is an array of z coordinate of each atom in ligand
    private int coSizeI; // This is a number of atoms in ligand

    //Constructor for Ligand Class
    //PreCondition: input: x, y, z Co-ordinate in floating point type
    //PostCondition: output: Parallel Array of x, y, z Co-ordinate of Ligand
    public Ligand(float[] xCoin, float[] yCoin, float[] zCoin) {
        coSizeI = xCoin.length;
        xCoF = new float[coSizeI];
        yCoF = new float[coSizeI];
        zCoF = new float[coSizeI];
        for (int i = 0; i < coSizeI; i++) {
            this.xCoF[i] = xCoin[i];
            this.yCoF[i] = yCoin[i];
            this.zCoF[i] = zCoin[i];
//          System.out.println(xCo[i] + " " + yCo[i] + " " + zCo[i]);
        }
    }

    //Copy-Constructor for Ligand Class
    //PreCondition: input: ligand
    //PostCondition: output: new ligand with Co-ordinate
    public Ligand(final Ligand lg) {
        this.coSizeI = lg.coSizeI;
        xCoF = new float[coSizeI];
        yCoF = new float[coSizeI];
        zCoF = new float[coSizeI];
        for (int i = 0; i < coSizeI; i++) {
            this.xCoF[i] = lg.xCoF[i];
            this.yCoF[i] = lg.yCoF[i];
            this.zCoF[i] = lg.zCoF[i];
//          System.out.println(xCo[i] + " " + yCo[i] + " " + zCo[i]);
        }
    }

    //This method returns the Number of Atoms in Ligand.
    public int numAtom() {
        return coSizeI;
    }

    //This method returns Co-ordinate of Atom according to axis (x,y,z) and
    //position in array
    public float coAtom(char axis, int order) {
        if (axis == 'x') {
            return this.xCoF[order];
        } else if (axis == 'y') {
            return this.yCoF[order];
        } else if (axis == 'z') {
            return this.zCoF[order];
        } else {
            System.out.println("Wrong Atom Type!!");
            return -0.0f;
        }
    }

    //Override println of Ligand Class
    public String toString() {
        DecimalFormat dF = new DecimalFormat("###0.000");
        String s = new String();
        StringBuffer sbf = new StringBuffer();
        for (int j = 0; j < coSizeI; j++) {
            sbf = sbf.append("CoPos[" + (j + 1) + "]: \t" + dF.format(xCoF[j])
                    + "\t\t" + dF.format(yCoF[j]) + "\t\t" + dF.format(zCoF[j])
                    + '\n');
        }
        s = sbf.toString();
        return s;
    }
}
```

Appendix E

Source Code of EnCo.java

```
public class EnCo implements Comparable {

    private float energyF = 0.0f;
    private int xI = 0;
    private int yI = 0;
    private int zI = 0;
    private int sphereIndex = 0;
    private int elementIndex = 0;
    //Constructor for EnCo:
    //input: energy and x, y, z Co-ordinate
    public EnCo(float enF, int zcoI, int ycoI, int xcoI) {
        this.energyF = enF;
        this.zI = zcoI;
        this.yI = ycoI;
        this.xI = xcoI;
    }

    //Constructor for EnCo:
    //input: energy and Sphericle angle position which encrypt in sphereIndex
    public EnCo(float enF, int spi)
    {
        this.energyF = enF;
        this.sphereIndex = spi;
    }

    //Constructor for EnCo:
    //input: energy and Sphericle angle which encrypt in sphereIndex and Element
index
```

```
public EnCo(float enF, int spi, int elmi)
{
   this.energyF = enF;
   this.sphereIndex = spi;
   this.elementIndex = elmi;
}

//return sphereIndex
public int getSpIndex()
{
   return this.sphereIndex;
}

//return ElementIndex
public int getElementIndex()
{
   return this.elementIndex;
}

//return Energy
public float getEnergyF() {
   return this.energyF;
}

//return x Co-ordinate
public int getXI() {
   return this.xI;
}

//return y Co-ordinate
public int getYI() {
   return this.yI;
```

```
    }

    //return z Co-ordinate
    public int getZI() {
        return this.zI;
    }

    //Override println of EnCo class
    public String toString() {
        String coEnString;
        coEnString = this.energyF + " " + this.zI + " " + this.yI + " " + this.xI + " " +
this.sphereIndex + " " + this.elementIndex;
        return coEnString;
    }

    //Overide Comparable Class
    public int compareTo(Object o) {
        if (o instanceof EnCo) {
            EnCo x = (EnCo) o;
            return (int) (this.energyF * 1000) - (int) (x.energyF * 1000);
        }
        throw new RuntimeException();
    }

}
```

Appendix F

Source Code of CUDockUtil.java

```
public class CUDockUtil implements MapConstant {

    //Static Method for Loading File from File Location
    //input: file location
    //out: String in which that file contains.
    //exception: IOException when that file can not open
    public static String loadFile(String fl)    //fl is file location
    {
        StringBuffer sfmap = new StringBuffer("");

        try {
            Scanner in = new Scanner(new File(fl)); // for example
"C:/Users/admin/Desktop/JEclipseWorkSpace/MapMaker/src/hsg1_rigid.C.map"

            while (in.hasNext()) {
                String line = in.nextLine();
                line = line.trim();
                if (line.length() > 0) {
                    sfmap = sfmap.append(line + "\n");
                }

            }
            in.close();
        } catch (IOException e) {
            System.out.println("Error opening input file ");//+fl
            System.exit(0);
        }
        return sfmap.toString();
```

```
}

//Static Method for Generating File from String
//input: String that need to print out in File and
//The location of the output File
//output: File that contains input String in output file location
//Exception: IOException when the output file can not create
public static void genFileFromString(String sp, String outFile)
{
   try {
      PrintStream out = new PrintStream(new File(outFile));
      StringBuffer aBuffer = new StringBuffer();
      aBuffer.append(sp);
      out.println(aBuffer.toString());

      out.close();
   } catch (IOException e) {
      System.out.println("Error opening output file ");//+fl
      System.exit(0);
   }
}

//Static Method for generating PDB Map
//input: Ligand, Template Map, Location of output file
//output: PDB Map File with new co-ordinate at the output file location
//Exception: IOException when the output file can not create.
public static void genPDBMap(Ligand lg, String tempMap, String outFile)
{
   try {
      PrintStream out = new PrintStream(new File(outFile));
      StringBuffer aBuffer = new StringBuffer();
      aBuffer.append(tempMap);
```

```
        int pxs = pdbXCoS - 1;
        int pys = pdbYCoS - 1;
        int pzs = pdbZCoS - 1;
        int pxe = pdbXCoE;
        int pye = pdbYCoE;
        int pze = pdbZCoE;

        for (int i = 0; i < lg.numAtom(); i++) {
            aBuffer.replace(pxs, pxe, eightDigit(lg.coAtom('x', i)));
            aBuffer.replace(pys, pye, eightDigit(lg.coAtom('y', i)));
            aBuffer.replace(pzs, pze, eightDigit(lg.coAtom('z', i)));
            pxs += nextCo;
            pys += nextCo;
            pzs += nextCo;
            pxe += nextCo;
            pye += nextCo;
            pze += nextCo;
        }
        out.println(aBuffer.toString());
        out.close();
    } catch (IOException e) {
        System.out.println("Error opening output file ");//+fl
        System.exit(0);
    }
}
//Static Method for Formating input(type double) into ####.### form
//input: Number in type double
//output: String of formated Number in form ####.###
public static String eightDigit(double d) {
    DecimalFormat dF = new DecimalFormat("###0.000");
    String s = "        " + dF.format(d);
    return s.substring(s.length() - 8, s.length());
```

```
}
//sin method
//input: degree
//output: The value of sin of that degree
public static double sin(double degree) {
   return Math.sin(Math.toRadians(degree));
}
//cos method
//input: degree
//output: The value of cos of that degree
public static double cos(double degree) {
   return Math.cos(Math.toRadians(degree));
}

//Method for read integer from message
//input: Scanner and integer number in String type
//output: integer of that String in integer type
public static int readInt(Scanner sc, String msg) {
   int v = 0;
   boolean success = false;
   do {
      try {
         if (!"".equals(msg)) {
            System.out.print(msg);
         }
         v = sc.nextInt();
         success = true;
      } catch (InputMismatchException e) {
         if (!"".equals(msg)) {
            System.out.println("To be corrected, please input INTEGER");
         }
         sc.next();
```

```
            }
        } while (!success);
        return v;
    }

    //Method for read Double from message
    //input: Scanner and double number in String type
    //output: integer of that String in double type
    public static double readDouble(Scanner sc, String msg) {
        double v = 0;
        boolean success = false;
        do {
            try {
                if (!"".equals(msg)) {
                    System.out.print(msg);
                }
                v = sc.nextDouble();
                success = true;
            } catch (InputMismatchException e) {
                if (!"".equals(msg)) {
                    System.out.println("To be corrected, please input DOUBLE");
                }
                sc.next();
            }
        } while (!success);
        return v;
    }
}
```

Appendix G

Source Code of MapConstant.java

```
public interface MapConstant {

    public static final int nextCo   = 79;
    public static final int pdbXCoS  = 31;
    public static final int pdbXCoE  = 38;
    public static final int pdbYCoS  = 39;
    public static final int pdbYCoE  = 46;
    public static final int pdbZCoS  = 47;
    public static final int pdbZCoE  = 54;
    public static final int elemSymS = 77;
    public static final int elemSymE = 79;
    public static final int chargeS  = 70;
    public static final int chargeE  = 76;

    public static final float sin0  = 0.0f;
    public static final float cos0  = 1.0f;
    public static final float sin45 = (float)Math.sin(Math.toRadians(45));
    public static final float cos45 = (float)Math.cos(Math.toRadians(45));
    public static final float sin90 = 1.0f;
    public static final float cos90 = 0.0f;
    public static final float sin135 = (float)Math.sin(Math.toRadians(135));
    public static final float cos135 = (float)Math.cos(Math.toRadians(135));
    public static final float sin180 = 0.0f;
    public static final float cos180 = -1.0f;
    public static final float sin225 = (float)Math.sin(Math.toRadians(225));
    public static final float cos225 = (float)Math.cos(Math.toRadians(225));
    public static final float sin270 = -1.0f;
    public static final float cos270 = 0.0f;
    public static final float sin315 = (float)Math.sin(Math.toRadians(315));
    public static final float cos315 = (float)Math.cos(Math.toRadians(315));
    public static final float sin360 = 0.0f;
    public static final float cos360 = (float)Math.cos(Math.toRadians(360));

}
```

Appendix H

Source Code of MatrixUtil.java

```
// transpose Matrix
// input: A
// keep output in: AT
public static void transpose(final float[][] A, float[][] AT) {
    for (int i = 0; i < matDim; i++) {
        for (int j = 0; j < matDim; j++) {
            AT[j][i] = A[i][j];
        }
    }
}
// multiply Matrix
// input: A * B
// keep output in: C
public static void multiply(final float[][] A, final float[][] B, float[][] R) {
    for (int i = 0; i < matDim; i++) {
        float[] aRowi = A[i];
        float[] rRowi = R[i];
        for (int k = 0; k < matDim; k++) {
            float[] bRowk = B[k];
            float aik = aRowi[k];
            for (int j = 0; j < matDim; j++) {
                rRowi[j] += aik * bRowk[j];
            }
        }
    }
}
// inverse Matrix
// input: A
```

```
    // keep output in: R
    public static void inverse(final float[][] A, float[][] R) {
        int n = A.length;
        float x[][] = R;//new double[n][n];
        float b[][] = new float[n][n];
        int index[] = new int[n];
        // Initiate the matrix identity
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i == j) {
                    b[i][i] = 1;
                } /*else {      //deleting this part because default is 0
                 b[i][j] = 0;
                 }*/
            }
        }
//        float[][] tempA = A;
        // Transform the matrix into an upper triangle
        gaussian(A, index);
        // Update the matrix b[i][j] with the ratios stored
        for (int i = 0; i < n - 1; ++i) {
            for (int j = i + 1; j < n; ++j) {
                for (int k = 0; k < n; ++k) {
                    b[index[j]][k] -= A[index[j]][i] * b[index[i]][k];
                }
            }
        }
        // Perform backward substitutions
        for (int i = 0; i < n; ++i) {
            x[n - 1][i] = b[index[n - 1]][i] / A[index[n - 1]][n - 1];
            for (int j = n - 2; j >= 0; --j) {
                x[j][i] = b[index[j]][i];
            for (int k = j + 1; k < n; ++k) {
                x[j][i] -= A[index[j]][k] * x[k][i];
            }
            x[j][i] /= A[index[j]][j];
        }
    }
}
// Method to carry out the partial-pivoting Gaussian elimination.
// Here index[] stores pivoting order.
private static void gaussian(float a[][], int index[]) {
    int n = index.length;
    float c[] = new float[n];
    // Initialize the index
    for (int i = 0; i < n; ++i) {
        index[i] = i;
    }
    // Find the rescaling factors, one from each row
    for (int i = 0; i < n; ++i) {
        float c1 = 0;
        for (int j = 0; j < n; ++j) {
            float c0 = Math.abs(a[i][j]);
            if (c0 > c1) {
                c1 = c0;
            }
        }
        c[i] = c1;
    }
    // Search the pivoting element from each column
    int k = 0;
    for (int j = 0; j < n - 1; ++j) {
        float pi1 = 0;
        for (int i = j; i < n; ++i) {
                float pi0 = Math.abs(a[index[i]][j]);
                pi0 /= c[index[i]];
                if (pi0 > pi1) {
                    pi1 = pi0;
                    k = i;
                }
            }
            // Interchange rows according to the pivoting order
            int itmp = index[j];
            index[j] = index[k];
            index[k] = itmp;
            for (int i = j + 1; i < n; ++i) {
                float pj = a[index[i]][j] / a[index[j]][j];
                // Record pivoting ratios below the diagonal
                a[index[i]][j] = pj;
                // Modify other elements accordingly
                for (int l = j + 1; l < n; ++l) {
                    a[index[i]][l] -= pj * a[index[j]][l];
                }
            }
        }
    }
    // x, y, z is the starting point of ligand that need to Rotate.
    // Cx, Cy, Cz is the end point of ligand part that rotate.
    // R is 3 dimensions Array: [0-71][4][4] 0-72 is angle, and 4*4 is transfermatrix
    public static void turnAroundBy5AtOriginV1(float Cx, float Cy, float Cz,
float[][][] R) {
        float phi = 0.f;
        float temp1[][] = new float[4][4];
        float temp2[][] = new float[4][4];
        float tempInverse[][] = new float[4][4];
        float temp3[][] = new float[4][4];
float d = (float) Math.sqrt((double) Cy * Cy + (double) Cz * Cz);
if (d == 0) {
    System.out.println("No length");
    return;
}
float d1 = (float) Math.sqrt((double) Cx * Cx + (double) Cz * Cz);
if (d1 == 0) {
    System.out.println("No length");
    return;
}
float[][] Rx = {{1.f, 0.f, 0.f, 0.f},
    {0.f, Cz / d, -(Cy / d), 0.f},
    {0.f, Cy / d, Cz / d, 0.f},
    {0.f, 0.f, 0.f, 1.f}
};
float[][] Ry = {{Cz/d1, 0.f, Cx/d1, 0.f},
    {0.f, 1.f, 0.f, 0.f},
    {-Cx/d1, 0.f, Cz/d1, 0.f},
    {0.f, 0.f, 0.f, 1.f}
};
MatrixUtil.multiply(Rx, Ry, temp2);
float tempR[][] = new float[4][4];
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        tempR[i][j] = temp2[i][j];
    }
}
MatrixUtil.inverse(tempR, tempInverse);
for (int i = 0; i < 72; i++) {
    float[][] Rz = new float[4][4];
    if (phi == 0.f) {
        float[][] Rzz = {{cos0, -sin0, 0, 0},
            {sin0, cos0, 0, 0},
            {0, 0, 1, 0},
            {0, 0, 0, 1}
        };
        Rz = Rzz;
    } else if (phi == 90.f) {
        float[][] Rzz = {{cos90, -sin90, 0, 0},
            {sin90, cos90, 0, 0},
            {0, 0, 1, 0},
            {0, 0, 0, 1}
        };
        Rz = Rzz;
    } else if (phi == 180.f) {
        float[][] Rzz = {{cos180, -sin180, 0, 0},
            {sin180, cos180, 0, 0},
            {0, 0, 1, 0},
            {0, 0, 0, 1}
        };
        Rz = Rzz;
    } else if (phi == 270.f) {
        float[][] Rzz = {{cos270, -sin270, 0, 0},
            {sin270, cos270, 0, 0},
            {0, 0, 1, 0},
            {0, 0, 0, 1}
        };
        Rz = Rzz;
    } else if (phi == 360.f) {
        float[][] Rzz = {{cos360, -sin360, 0, 0},
            {sin360, cos360, 0, 0},
            {0, 0, 1, 0},
            {0, 0, 0, 1}
        };
```

```
                Rz = Rzz;
            } else {
                float[][] Rzz = {{(float) Math.cos((double) phi), (float) -Math.sin((double)
phi), 0, 0},
                    {(float) Math.sin((double) phi), (float) Math.cos((double) phi), 0, 0},
                    {0, 0, 1, 0},
                    {0, 0, 0, 1}
                };
                Rz = Rzz;
            }
            MatrixUtil.multiply(temp2, Rz, temp3);
            MatrixUtil.multiply(temp3, tempInverse, R[i]);
            temp3 = new float[4][4];
            phi += 5.f;
        }
    }
```

BIOGRAPHY

Mr. Thotsaphon Bowornthanitkun received B.Sc. in Food Science and Technology at Kasetsart University, Thailand, in 1997. After graduation, he worked as a chemist in an edible oil company. A year later, Thotsaphon went to work for NC. Enterprise company and study at California State University, Long Beach, USA, where he got a Certificate in Computer Science and Engineering. In 2005, he obtained M.Sc. in Computer Science at Thammasat University, Thailand. After graduation, he had been working in a position of wireless software engineer at a British multinational company.