The Development of High Performance Structure and Ligand Based Virtual Screening Techniques

Thesis Submitted for the Degree of

Doctor of Philosophy

Steven R. Shave BSc. MSc.

Structural Biochemistry Group

The Institute of Structural and Molecular Biology

University of Edinburgh

August 2009

Abstract

Virtual Sreening (VS) is an in silico technique for drug discovery. An overview of VS methods is given and is seen to be approachable from two sides: structure based and ligand based. Structure based virtual screening uses explicit knowledge of the target receptor to suggest candidate receptor-ligand complexes. Ligand based virtual screening can infer required characteristics of binders from known ligands. A consideration for all virtual screening techniques is the amount of computing time required to arrive at a solution. For this reason, techniques of high performance computing have been applied to both the structural and ligand based approaches.

A proven structure based virtual screening code LIDAEUS (Ligand Discovery At Edinburgh UniverSity) has been ported and parallelised to a massively parallel computing platform, the University of Edinburgh's IBM Bluegene/l, consisting of 2,048 processor cores. A challenge in achieving scaling to such a large number of processors required implementation of a minimal communication parallel sort algorithm. Parallel efficiencies achieved within this parallelisation exceeded 99%, confirming that a near optimum strategy has been followed and capacity for running the code on a greater number of processors exists. This implementation of the program has been successfully used with a number of protein targets.

The development of a new ligand based virtual screening code has been completed. The program UFSRAT (Ultra Fast Shape Recognition with Atom Types) takes the

features of known binders and suggests molecules which will be able to make similar interactions. This similarity method is both fast (1 million molecules per hour per processor) and independent of input orientation. Along with UFSRAT, some other methods (VolRAT and UFSRGraph) based on UFSRAT have been developed, addressing different approaches to ligand based virtual screening. UFSRAT as an approach to discovering novel protein-ligand complexes has been validated with the discovery of a number of inhibitors for 11β-Hydroxysteroid Dehydrogenase type 1 and FK binding protein 12.

Declaration

The work presented in this thesis is the original work of the author. This thesis has been composed by the author and has not been submitted in whole or in part for any other degree.

Steven Robert Shave

Acknowledgements

Firstly, I would like to thank my supervisors Prof. Malcolm Walkinshaw and Dr. Paul Taylor for their support, guidance, patience and encouragement during my studies.

Secondly, my thanks goes out to members of the Walkinshaw group, especially Kun-Yi Hsin, Liz Blackburn, Hugh Morgan, Simon Harding, Peter Brown and Wissam Mehio.

Thanks go to members of EPCC at the University of Edinburgh, and particularly Dr Lorna Smith for her help and insights into High Performance Computing.

Finall I thank my family, Glenda, Robert, Stewart and Eve-Marie. Without either one the following document would not exist.

Abbreviation list

11ßHSD1	11ß-hydroxysteroid dehydrogenase type 1
11ßHSD2	11ß-hydroxysteroid dehydrogenase type 2
2D	2 Dimensional
3D	3 Dimensional
Å	Ångström
ADMET	Absorption, Distribution, Metabolism, Excretion, and Toxicity
BLAS	Basic Linear Algebra Subprograms
CLU	Cluster File
CNK	Compute Node Kernel
Co	Co-processor
CoMFA	Comparative Molecular Field Analysis
CPU	Central Processing Unit
CT	Chemical Table
Ctab	Connection Table
Da	Dalton
DMSO	Dimethyl sulfoxide
DNA	Deoxyribonucleic acid
EDULISS	Edinburgh University Ligand Selection System
ESSL	IBM Engineering and Scientific Subroutine Library
FFTW	Fastest Fourier Transform in the West
FKBP12	FK Binding Protein 12
FLOPS	Floating Point Operations Per Second
FORTRAN	Formula Translating System
FPU	Floating Point Unit
GB	GigaBytes
GCC	GNU Compiler Collection
GHz	Gigahertz
GNU	GNU's Not Unix
GROMACS	Groningen Machine for Chemical Simulations
HECToR	High-End Computing Terascale Resource
HIV	Human immunodeficiency virus
HPC	High Performance Computing
HTVS	High Throughput Virtual Screening
IBM	International Business Machines
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers

IO	Input/Output
JIT	Just In Time
kcal	thousand calories
kcal/mol	Kilo calories per mol
LIDAEUS	Ligand Discovert at Edinburgh University
LINUX	LINUX Is Not Unix
LU	Lower/Upper
M	Million
MFLOPS	Mega Floating Point Operations Per Second
MHz	MegaHertz
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MOL	MDL Mol File
mol	mole
MPI	Message Passing Interface
NADP/NADPH	Nicotinamide Adenine Dinucleotide Phosphate
NAMD	Nanoscale Molecular Dynamics
NMR	Nuclear Magnetic Resonance
openMosix	open Multicomputer Operating System for Unix
OpenMP	Open Multi-processing
PDB	Protein Data Bank
PDB ID	Protein Data Bank Identifier
PMF	Potentials of Mean Force
QCDOC	Quantum Chromodynamics on a Chip
QSAR	Quantitative Structure-Activity Relationship
RAM	Random Access Memory
RCSB	Research Collaboratory for Structural Bioinformatics
RMSD	Root Mean Squared Deviation
RNA	Ribonucleic acid
RO3	Rule of 3
RO5	Rule of 5
SAR	Structure-Activity Relationship
SCOP	Structural Classification of Proteins
SDF	Structure-Data File
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SL	Steven Shave/Liz Blackburn
SMILES	Simplified molecular input line entry specification

SPA	Scintillation proximity assay
TDF	Thermal denaturation fluorescence
UFSR	Ultrafast Shape Recognition
UFSRAT	Ultrafast Shape Recognition with Atom Types
UFSRGraph	Ultrafast Shape Recognition from Graph Decomposition
vdW	van der Waals
VN	Virtual node
VolRat	Volumetric Recognition with Atom Types
VS	Virtual Screening
ZINC	ZINC Is Not Commercial

Table of Contents

Table of Figures

1 Introduction

1.1 Foreword

With the advent of the microprocessor and integrated circuit, the prospects for researchers to experiment in a virtual laboratory have grown to fruition. This is seen in almost all fields of research; simulation being considered a technique which is created from, often precedes and runs parallel with real world or laboratory experimentation. The microprocessors used in modern computing platforms have increased almost exponentially in their features, speed and reliability over the last decades. Coupled with the ever expanding ability to store and manage data, simulation has been adopted as almost a standard practice to aid real world experimentation, visualise and interpret results. The ability for simulation to run in many stages alongside real world experimentation enables preliminary hypotheses to be explored rapidly without the associated costs of real world experiments. Simulation may also help guide experimentation, allowing further exploration of research options. Areas also opened up by simulation allow the analysis of situations (systems) where real world experimentation may be too small, large, costly or dangerous.

Whilst the above text introduces simulation as almost a necessity to modern science, it is worth noting the many pitfalls present. Any simulation study carried out is being performed on a model of the system, not the system itself. Ideally the model will be such a good mimic of the system that a simulation truth will also be a system

truth. This is not always the case, and is the major caveat. Model errors can be present in any and multiple levels of the design; a simplification built up of conceptual layers representing the target system. The first layer is a mathematical model which is thought to mimic the system. This is then discretised, turning a continuous and infinitely fine model into a manageable representation. Mathematical methods are then applied to this discretised representation of the system, creating a computer code to run the simulation. The simulation design process viewed as layers shows how easily incorrect decisions can result in a simulation which does not properly mimic reality. The care with which each step must be carried out is extraordinary, with testing required at every step of the way. The difficulty in expressing real world systems in an abstracted form is summed up well in the following quote.

> "*As far as the laws of mathematics refer to reality, they are not certain, and as far as they are certain, they do not refer to reality*" – Albert Einstein

With the large caveat that the simulation of the system may not behave in the same way as the real system, simulation must go hand in hand with real world experiments, acting as an aid rather than a replacement. There is a danger that sole reliance upon a simulation may become, over time, a study of the complex simulation rather than the original target system. This may rapidly become the case due to the level of trust put into new technologies. This is illustrated in a quote by Charles Babbage, when presenting his groundbreaking 'difference engine' which was the first programmable computer made of clockwork and operated by hand.

> "*On two occasions I have been asked, 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' I am not able to rightly apprehend the kind of confusion that could provoke such a question*". – Charles Babbage [1].

It is important that simulation exists to guide real world experimentation and not replace it.

1.2 Molecular simulation

Coupled with the use of computational tools, molecular simulation has gained huge popularity. Molecular simulation is the using of tools which mimic properties of real atoms or larger chemical units such as amino acids. In this thesis, the main focus and use of molecular simulation has been to model the behaviour of atoms within either small molecules or larger macromolecules. However, molecular simulation methods can vary in their exhaustiveness and accuracy.

1.3 Virtual Screening

Virtual screening is an encompassing term [2] which falls under the broader category of molecular simulation. Virtual screening as used in the context of this thesis aims to find small molecule binders for a protein target (protein-ligand binding). It is then hoped that these binders will be present in a key location offering the researcher some level of control over the protein. Virtual screening is defined by Walters in the following paragraph:

> *"For the computational chemist, a laudable goal is to develop some kind of computer program capable of automatically evaluating large libraries of compounds. This process is a natural extension of what molecular modelling scientists currently do, albeit on fewer compounds and in a less automated way. This process is sometimes called 'virtual screening'".* [3]

As shown by Figure 1.1, virtual screening can be further broken down into structure based and ligand based virtual screening. Ligand based virtual screening uses known ligands to search for similar potential ligands in a database of potential ligands; no information as to the receptor which they bind is expressly encoded. The more common structure based virtual screening uses knowledge of the receptor and a database of candidate ligands to predict binding potential. Breaking structure based virtual screening down; we find two further branches, capacity driven and accuracy driven. A challenging aspect of virtual screening is to effectively screen a large number of potential ligands. However, a small number of potential ligands may be evaluated very accurately using accuracy driven approaches, such as quantum mechanical methods [4; 5; 6]. This report focuses on both virtual screening through both ligand based and capacity driven structure based approaches. Striving for capacity and the ability to evaluate the docking of millions of compounds in a single run allows the simulation to guide real world experimentation, using this for validation rather than highly accurate but slow simulations.

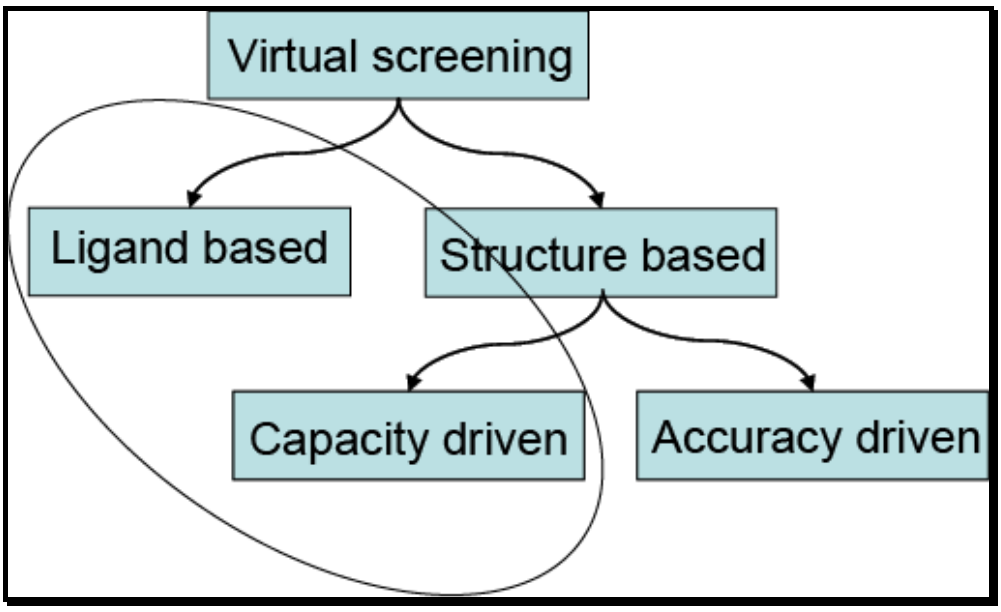


Figure 1.1 – Virtual screening decomposed. This report is concerned mainly with capacity driven, structure based virtual screening and ligand based virtual screening as circled.

A review of existing ligand-based techniques and a newly developed technique for ligand-based virtual screening (UFSRAT – Ultra Fast Shape Recognition with Atom Types) is available in chapter 5 of this report. Chapter 4 documents the in-house virtual screening code LIDAEUS and its parallelisation, which has allowed massively parallel runs to take place, routinely screening millions of compounds against a receptor. Chapter 4 also profiles a variety of other structure based virtual screening programs.

The increase in computing power over the last decades has enabled simulations to be run on protein-ligand binding. Simulating this binding is a major advantage to the researcher, as previously expert knowledge could be used and an assessment made as to whether a compound should be tested in the laboratory, using an appropriate assay to see if binding or inhibition of the protein occurred. This process was very time

consuming. Pre-screening a library of compounds first using virtual screening can act as a filter and directional aid.

1.3.1 Interactions of interest

For a virtual screening program to be able to place ligands in complex with a receptor it must be able to determine energetically favourable states. There are three distinct and high level energy terms that come into play when creating a protein-ligand complex. These are intramolecular forces, intermolecular forces and the displacement of solvent. The displacement of solvent can be seen as belonging to the intermolecular forces group, but here, it is kept separate for clarity.

Intramolecular forces

Intramolecular forces denote energies found within a molecule, such as energies required to stretch or compress bond lengths from ideal energy minima or deviation from ideal torsion angle minima. Section 2.5.4 Force field energy calculation and Section 4.5 Simple flexible docking in LIDAEUS deals with this topic in detail.

Intermolecular forces

Intermolecular forces are energies exerted between molecules. These energies come from three effects, the first being dipole moment.

Two molecules are shown in Figure 1.2 which give rise to a dipole moment and therefore a weak bonding effect is present.

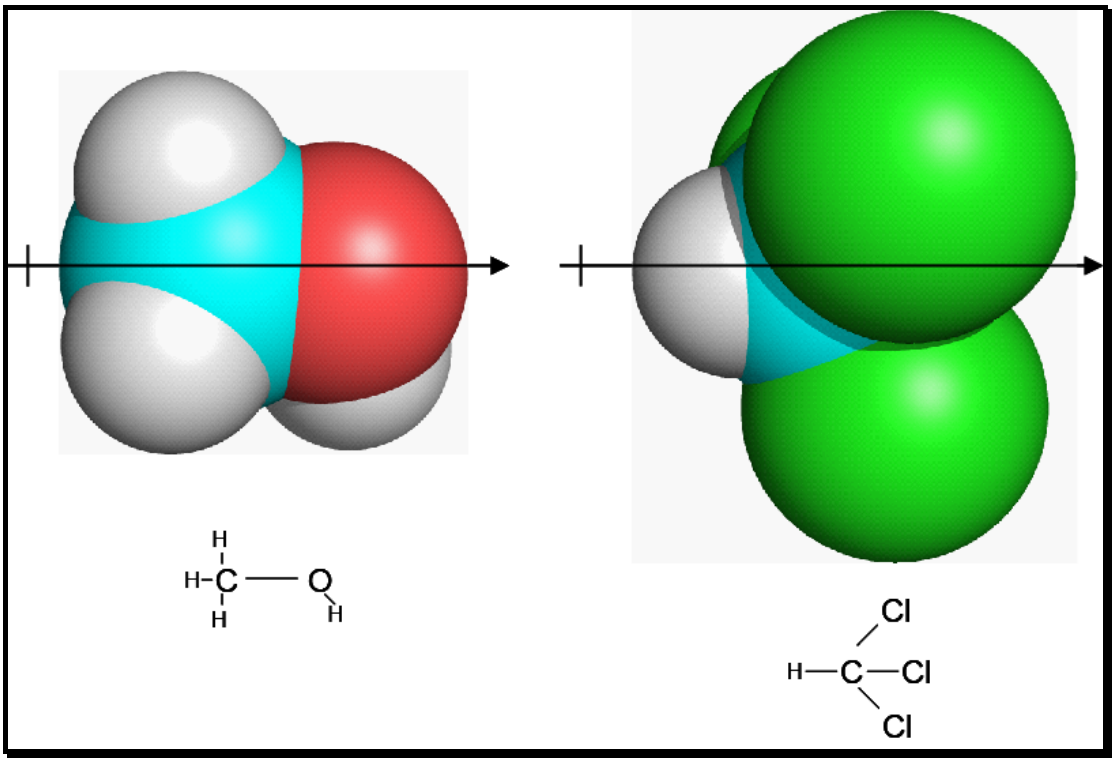


Figure 1.2 - Dipole-dipole interaction between methanol (left) and chloroform (right). 2D molecule diagram also given. The oxygen atom belonging to the methanol is slightly negative, electrons spending more time at the right hand side of the molecule. For the chloroform molecule, the chlorine atoms produce a negative charge on the right hand side. Carbon atoms shown as cyan, oxygen atoms as red, chlorine atoms as green and hydrogen atoms as grey.

The non symmetric nature of the molecules in Figure 1.2 creates a dipole-dipole interaction effectively bonding (weakly) the molecules together. The partially negatively charged oxygen atom makes the right hand side of the methanol molecule

in Figure 1.2 slightly negative, whilst the chlorine atoms make the right hand side of the chloroform molecule slightly negative and in turn the left hand side slightly positive. Between the two molecules exists a slightly positive and slightly negative area of space, and therefore there is an attractive force between the two molecules.

A molecule with a dipole moment can induce a dipole in a molecule that does not have one.

The second force is van der Waals (sometimes known as the London dispersion force). It is similar to an induced dipole, but this time the force can occur between two molecules without dipole moments. In a situation where two gaseous helium atoms are in close proximity to each other, the electrons in each atom's s orbital is evenly spaced when averaged over time. However, within a short enough time frame, it could be considered that the electrons are occupying one face of the atom. When a complimentary effect occurs on both atoms, an attractive force is applied to the atoms.

The third force is known as hydrogen bonding. When hydrogen is bound to a negatively charged atom such as oxygen or nitrogen within the molecule, this creates a partially charged area which acts like a dipole. The hydrogen is in effect partially positively charged and can form a relatively strong interaction with another negatively charged atom from a nearby molecule. An example of a hydrogen bond is given in Figure 1.3

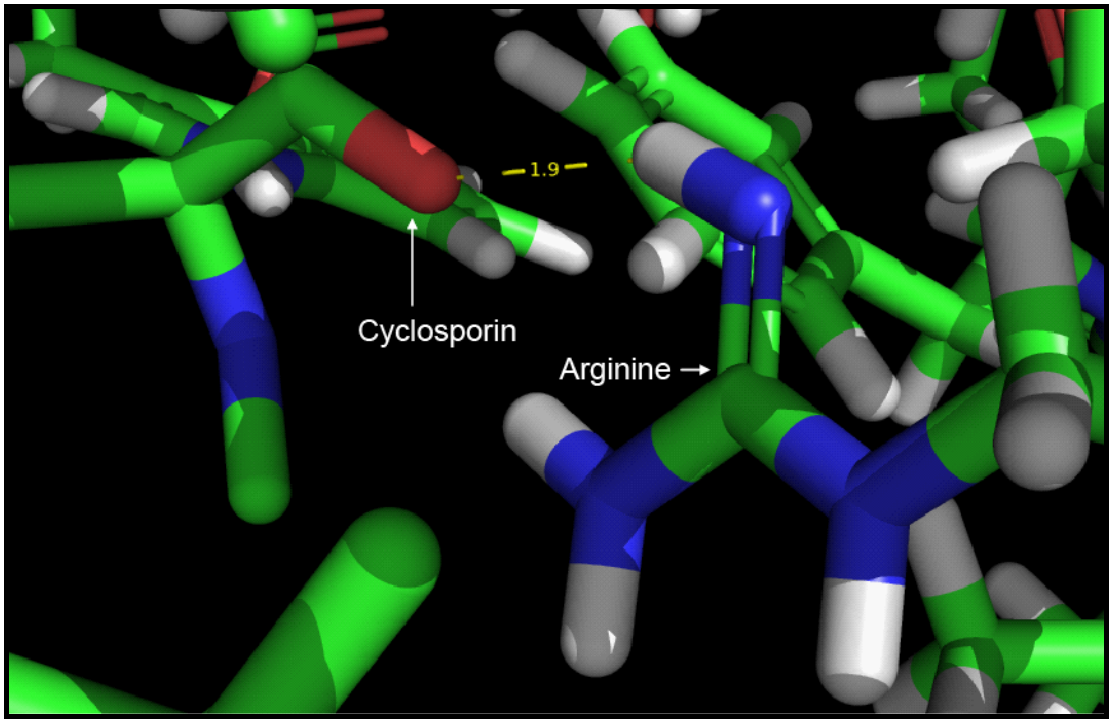


Figure 1.3 –An intermolecular hydrogen bond of length 1.9Å made between an acceptor oxygen atom of cyclosporin and an arginine residue from cyclophilin (complex represented from PDB with ID 1CWA). Oxygen atoms in red, nitrogen atoms in blue, carbon atoms in green and hydrogen atoms in grey.

As a general rule, the strongest intermolecular forces are hydrogen bonds, followed by dipole-dipole interactions and then the weakest being van der Waals.

1.3.2 Small molecule libraries and large macromolecule repositories

Walters postulates that virtual chemistry space contains perhaps around ten duotrigintillion, 1 googol or equivalently 10^{100} compounds [3]. Although this figure is a gross approximation, the scale of the possibilities is represented. Of this googol of chemical space, thankfully, only a small proportion is required for testing. Further discussion of suitable virtual screening libraries is given in chapter 3.

Alongside virtual screening exists *de novo* ligand design, where a biochemist uses tools or their expert knowledge to design a ligand capable of binding to a target. The newly created ligand is often an augmented version of a scaffold either common to known binders or known to bind itself. Computer tools designed to aid the biochemist may well use an ensemble of information from known binders [7; 8] as in the case of the Comparative Molecular Field Analysis (CoMFA) methodology [9; 10; 11; 12]. Schneider provides an overview of computational tools for *de novo* drug design [13; 14].

Structure based virtual screening is the act of using structural information about the receptor to guide the evaluation of potential ligands. Structural information on protein targets is widely available and structures are solved using a variety of techniques, the most common being X-ray diffraction and NMR (Nuclear Magnetic Resonance). A repository of structural information exists on proteins and other receptors. The Research Collaboratory for Structural Bioinformatics curates the Protein Data Bank [15; 16; 17] currently allowing the download of structural representations of over 55,000 proteins, nucleic acids and other structures. The resource is a free to use repository, which is the *de facto* standard place for the storing of structural bioinformatics information. With many structures being added each year by researchers, the number available has increased almost exponentially. The advent of new scientific methods and techniques means that receptors are usually represented many times as multiple entries, their structures being solved by a variety of ever more accurate techniques. For this reason, the figure of over 55,000

structures being available does not mean that each structure is unique as one may have been solved in complex with a natural ligand, another with a better more accurate (higher resolution technique) or with another method than is used as standard (X-ray diffraction versus NMR). Figure 1.4 shows the growth in holdings in recent years of the Protein Data Bank.

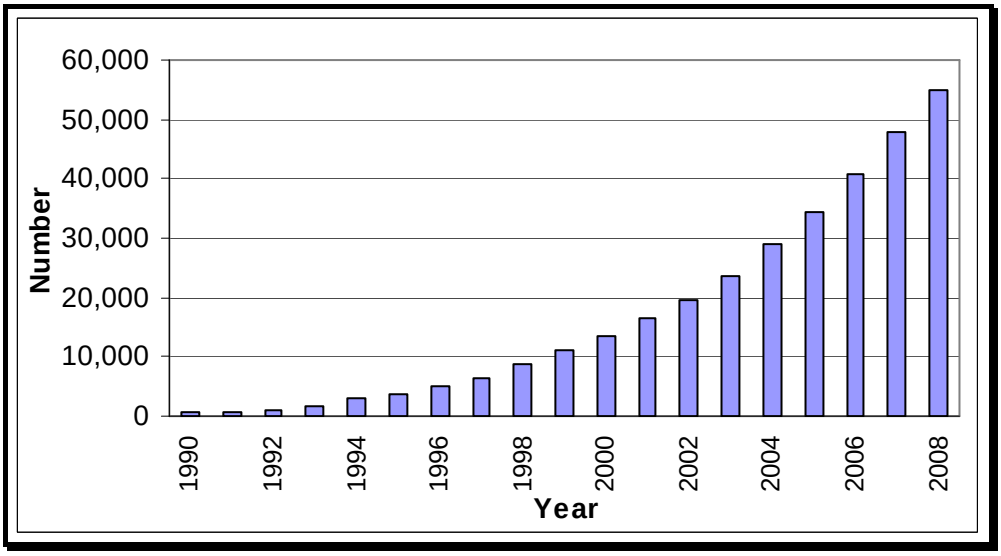


Figure 1.4 - Cumulative plot of Protein Data Bank holdings of structural information, data taken from: http://www.rcsb.org/pdb/statistics/contentGrowthChart.do?content=total&seqid=100 on 2nd January 2009.

Further statistical analysis of the Protein Data Bank reveals an interesting trend. With the way in which a protein folds into shape dictated by its primary structure, a program called SCOP (Structural Classification of Proteins) [18] has been used to identify unique folds present in the database. Using SCOP to identify unique folds in the database and discover how many new folds are added each year yields interesting results. Unique fold information is available from the PDB website.

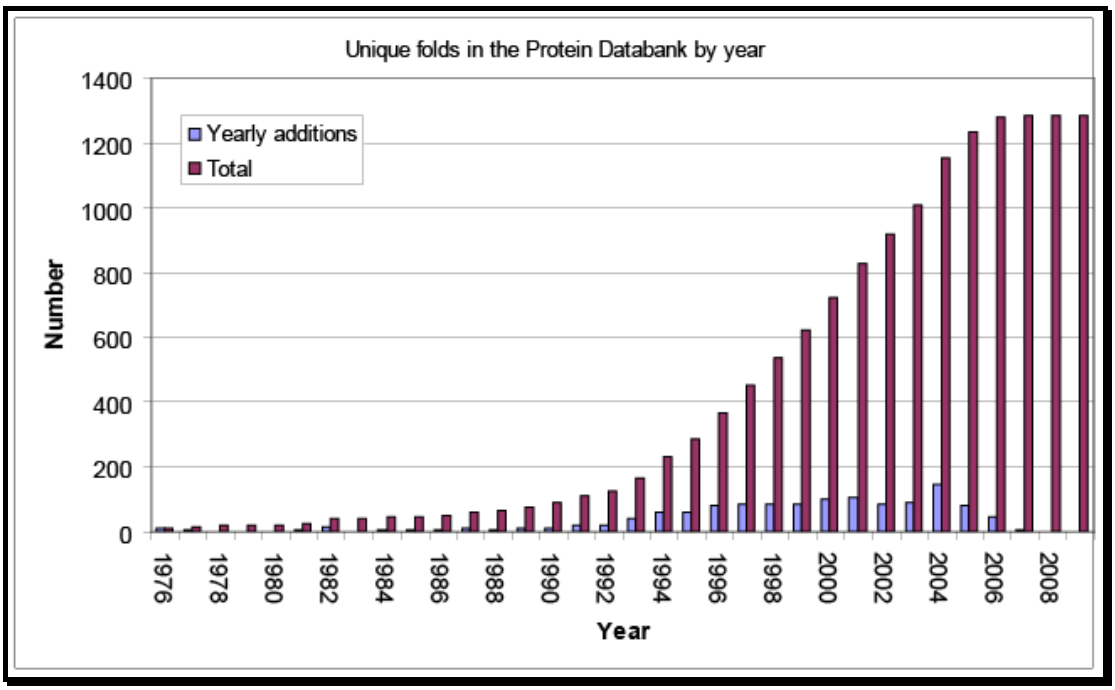


Figure 1.5 - Plot of unique folds present in Protein Datank holdings of structural information, data taken from http://www.rcsb.org/pdb/statistics/contentGrowthChart.do?content=fold-scop on 31st March 2009

Inspection of Figure 1.5 shows that the number of new, unique folds increased steadily up until 2004. After this date, although new folds are being added, the increase in numbers has drastically levelled off with fewer unique folds added each year. With the number decreasing, it could be argued that most unique folds for naturally occurring proteins of interest have been discovered.

Entries in the Protein Data Bank are assigned a unique identifier referred to here on as the Protein Database Identifier (PDB ID). Also, entries are given with their unique Digital Object Identifier (DOI). DOI is an online system for uniquely identifying online material. Information on DOI is available at: http://www.doi.org/

The massive amount of structural information available to researchers has made structure-based virtual screening a powerful approach. Within minutes of interest being expressed in a protein target, a researcher can have located many different structural representations of the protein and be viewing and exploring the structure. This structural information, once found to be fit for the purpose, can then be used in virtual screening runs.

1.3.3 Structure based virtual screening

A structure-based virtual screening run takes the following form: the protein of interest is examined and an interesting binding site or pocket found. If the location of the binding pocket is unknown, then many tools are available which use a variety of techniques to look for the most likely location for binding to occur such as Q-SiteFinder [19], Fuzzy Oil Drop [20] and LigProf [21]. The location of the binding site is then used as the point of interest upon or into which many candidate ligands will have their binding simulated. The simulation of binding typically returns a score for each candidate ligand in a specific location and conformation. The score given is often the enthalpy of binding given in kilo calories per mol (kcal/mol). From this score, a ranked list can be generated showing the best binders as predicted by the virtual screening run. Results from virtual screening can be used in a variety of ways; often the expert human knowledge of biologists and chemists is used to examine, pick out and extend scaffolds and molecules themselves in attempts to find better binders. Compounds predicted to bind well can then be taken forward and

tested in a biochemical laboratory using an appropriate assay for the target. A review of structure based virtual screening techniques is available in Chapter 4.

1.4 Protein targets of interest

The research documented in this report is primarily focused on the processes and algorithms used to facilitate virtual screening. However, any theoretical work must be backed up by experimental result. This work has focused on two proteins of interest; 11ß-hydroxysteroid dehydrogenase type 1 (from here on referred to as 11ßHSD1) and FK Binding Protein 12 (referred to as FKBP12). Research is ongoing within the department on these proteins, enabling collaboration and facilitating biochemical validation of predictions made during the course of research.

1.4.1 11ß-hydroxysteroid dehydrogenase type 1 (11ßHSD1)

While the conversion of cortisol to cortisone was first observed in 1953 by Amelung, it was only later that this activity was detected in a wide range of cell types.

Since then, the conversion has been of great interest to medicinal fields; 11ßHSD1 not only converting cortisone to cortisol but other glucocorticoids such as corticosterone to 11-dehydrocorticosterone [22]. The enzyme is bidirectional and dependant on cofactor availability although in a biologically relevant environment the cortisone to cortisol reaction is favoured.

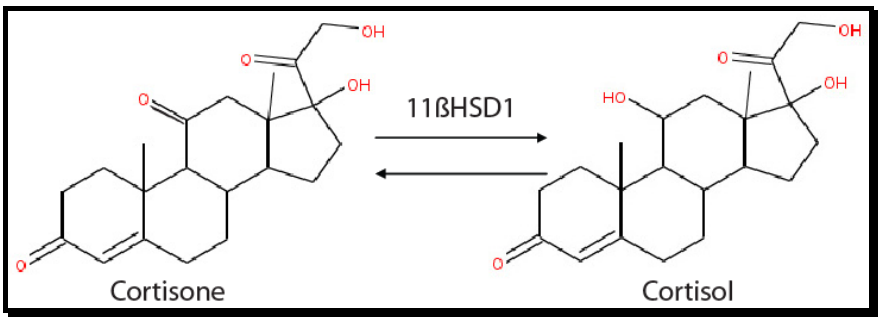


Figure 1.6 – Cortisone to Cortisol interconversion with 11ßHSD1

This research has only considered 11ßHSD1. The conversion of cortisone to cortisol is achieved through a co-factor which undergoes the conversion of NADPH to NADP. Cortisone and cortisol are shown in Figure 1.6

Over production of cortisol has been linked to many health issues such as obesity and "type two" diabetes in mice and humans [23]. It has been shown that inhibition of 11ßHSD1 leads to lower blood glucose levels in hyperglycaemic mice. This makes 11ßHSD1 a strong target for the treatment of type 2 diabetes [23; 24].

Glucocorticoids in general display links to obesity, diabetes, cardiovascular disease and also have an impact on memory function [25]. Controlling the rate at which cortisol is turned over into cortisone would be advantageous and this makes 11ßHSD1 a drug target of great importance. Published data shows a range of known inhibitors to 11ßHSD1, blocking the crucial binding site and disallowing the conversion of other glucocorticoids [25]. The attempts at virtual screening have focused on blocking the glucocorticoid binding site and not the NADP co-factor site.

The structural 11ßHSD1 data used during this course of research has been mainly from two entries in the Protein Data Bank. The entries have PDB IDs 2BEL and 2RBE.

In the Protein Data Bank, 11ßHSD1 is shown to crystallise as a homodimer. It has been demonstrated that 11ßHSD1 can exist as both a monomer [26] and a homodimer [27; 28]. There is evidence within crystal structures (Jillian Adie, personal communication, 2009) of a tetrameric arrangement which is also reported in the literature [29]. The tetrameric arrangement is reported to exist as a dimer/tetramer equilibrium in vitro, yet its occurrence *in vivo* is unclear [30].

The crystal structures with IDs 2BEL and 2RBE show the dimer in different states with the 2 units complexed in different conformations.

2BEL represents a homotetramer of 11ßHSD1 with NADP and carbenoxolone bound. (DOI 10.2210/pdb2bel/pdb)

2BEL is shown in Figure 1.7 with a Connolly surface [31; 32] applied and with a ligand bound in the active site in figure Figure 1.8.

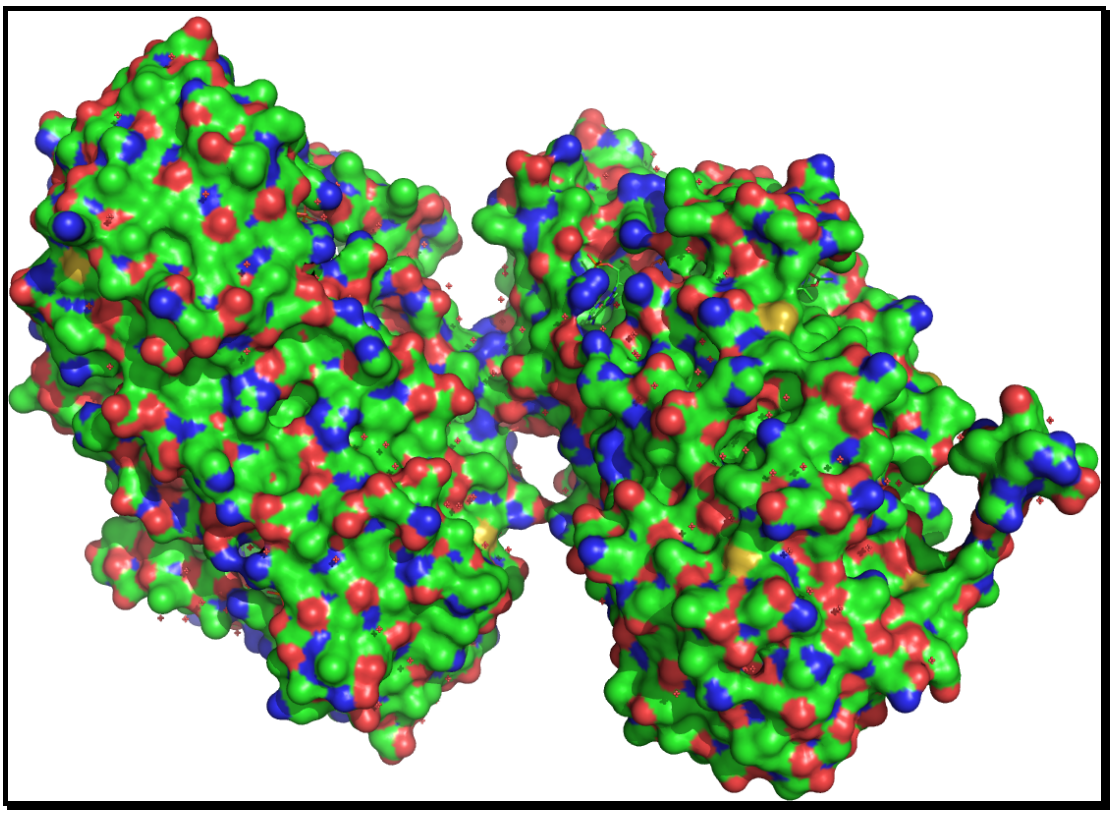


Figure 1.7 – Crystallographic dimer of 11ßHSD1 (PDBID 2BEL) shown with Connolly surface. Red crosses are crystallographic waters (the cross marks the position of the water's oxygen atom)

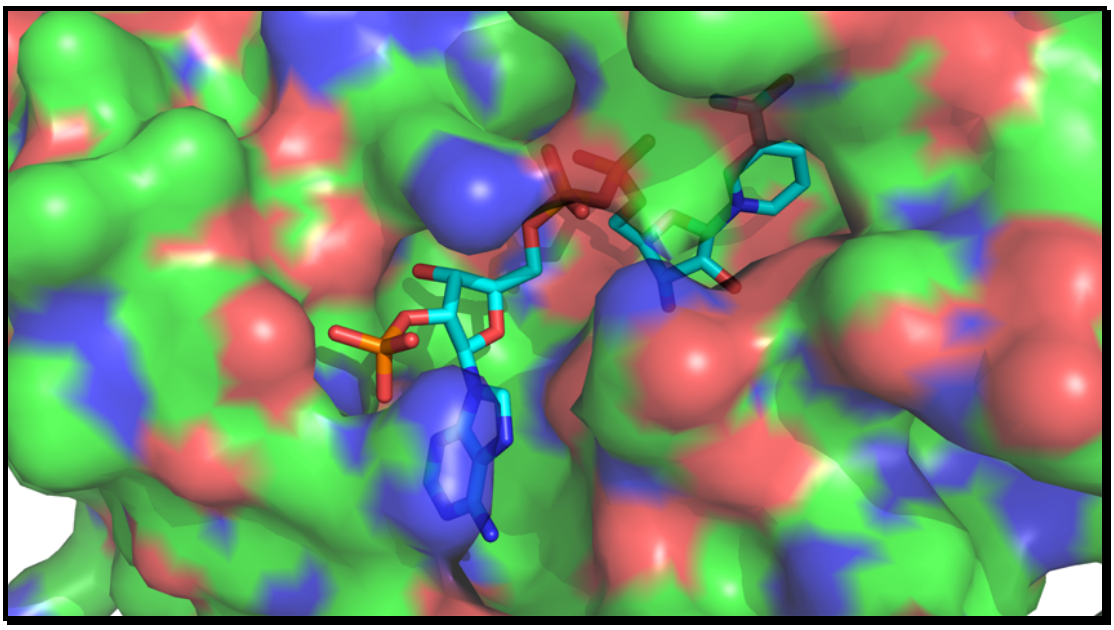


Figure 1.8 – NADP bound to 11ßHSD1 from the crystal structure 2BEL. Surface of 11ßHSD1 shown with carbon atoms as green, nitrogen as blue and oxygen as red. NADP shown as a stick representation. NADP carbon atoms shown as cyan, nitrogen as blue, oxygen as red, phosphorous as orange.

2RBE is a crystal structure achieved throughout research into 2-anilinathiazolone 11ßHSD1 inhibitors [33].

2RBE is shown in Figure 1.9 with a Connolly surface applied.

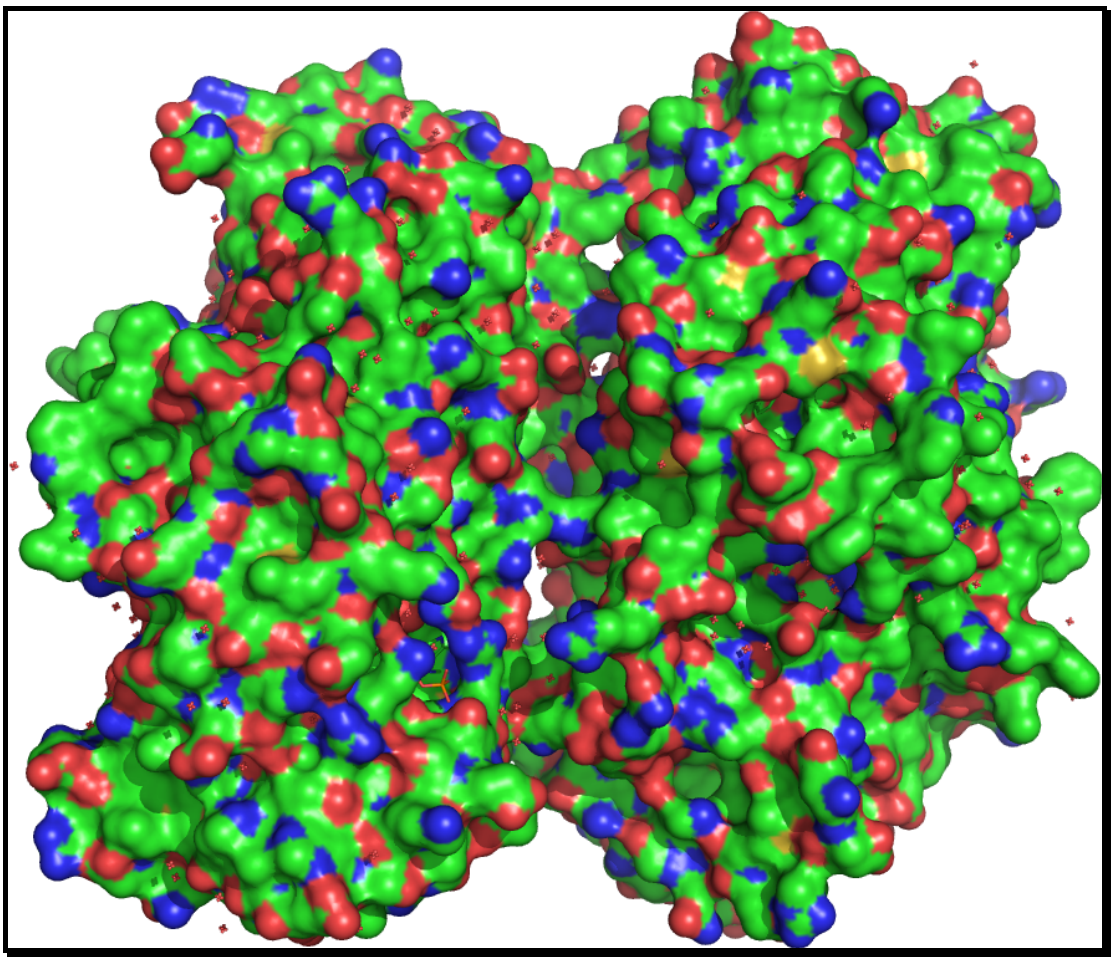


Figure 1.9 – Crystallographic dimer of 11ßHSD1 (PDBID 2RBE) shown with Connolly surface

1.4.2 FK506 Binding Protein 12 (FKBP12)

FK506 Binding Protein 12 (FKBP12) is an immunophilin found in many eukaryotes. FKBP12 readily complexes with the drug FK506 [34] (see Figure 1.11) , a large 'small molecule' of molecular weight 804 Da (see Figure 1.10). This FKBP12-FK506 complex acts as an immunosuppressive agent. The immunosuppressive properties are brought about by the complex inhibiting a transcription factor and ultimately stopping the release of T-cells; which cannot therefore induce death of cells not recognised by the immune system [35]. FK506 is often used in transplant therapy, suppressing the immune system and stopping rejection of transplanted organs [36]. Other immunosuppressive FKBP12 complexes are also known such as FKBP12-rapamycin which works on the same pathway (but at a different stage) to facilitate apoptosis [34].

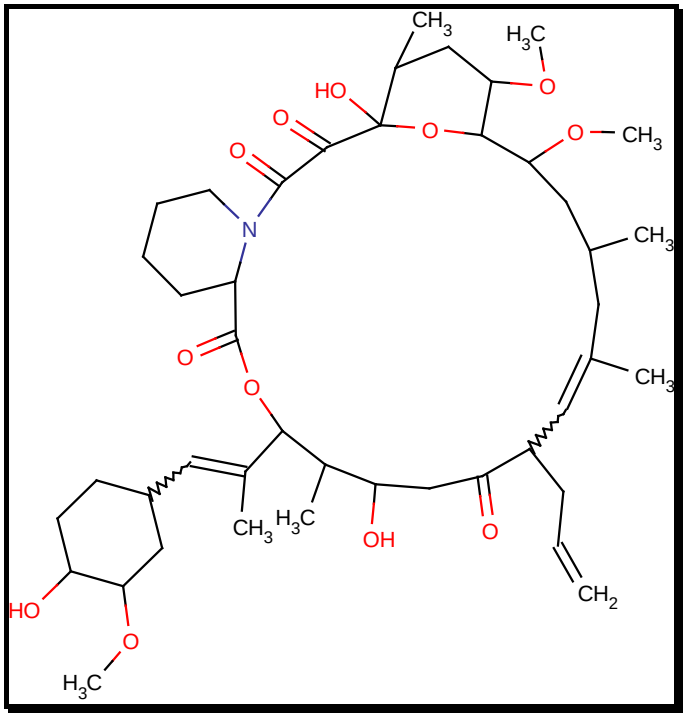


Figure 1.10 - 2D stick representation of FK506

Additionally, blocking of the active site of FKBP12 has been shown to be of importance in dealing with neurodegenerative conditions. Studies show that blocking of the FKBP12 active site has been useful in the treatment of Parkinson's disease in mouse models [37; 38; 39], making FKBP12 a target of prime interest for drug discovery.

The structural FKBP12 data used during this course of research has been mainly from two entries in the Protein Data Bank. These entries are PDBIDs 1FKJ (DOI 10.2210/pdb1fkj/pdb) and 2DG3 (DOI 10.2210/pdb2dg3/pdb). Both entries represent human FKBP12

1FKJ represents FKBP12 bound to FK506 and is the result of published work carried out by Wilson [34]. It is shown in Figure 1.11 with a Connolly surface applied to FKBP12 and a stick representation of FK506 completing the complex.

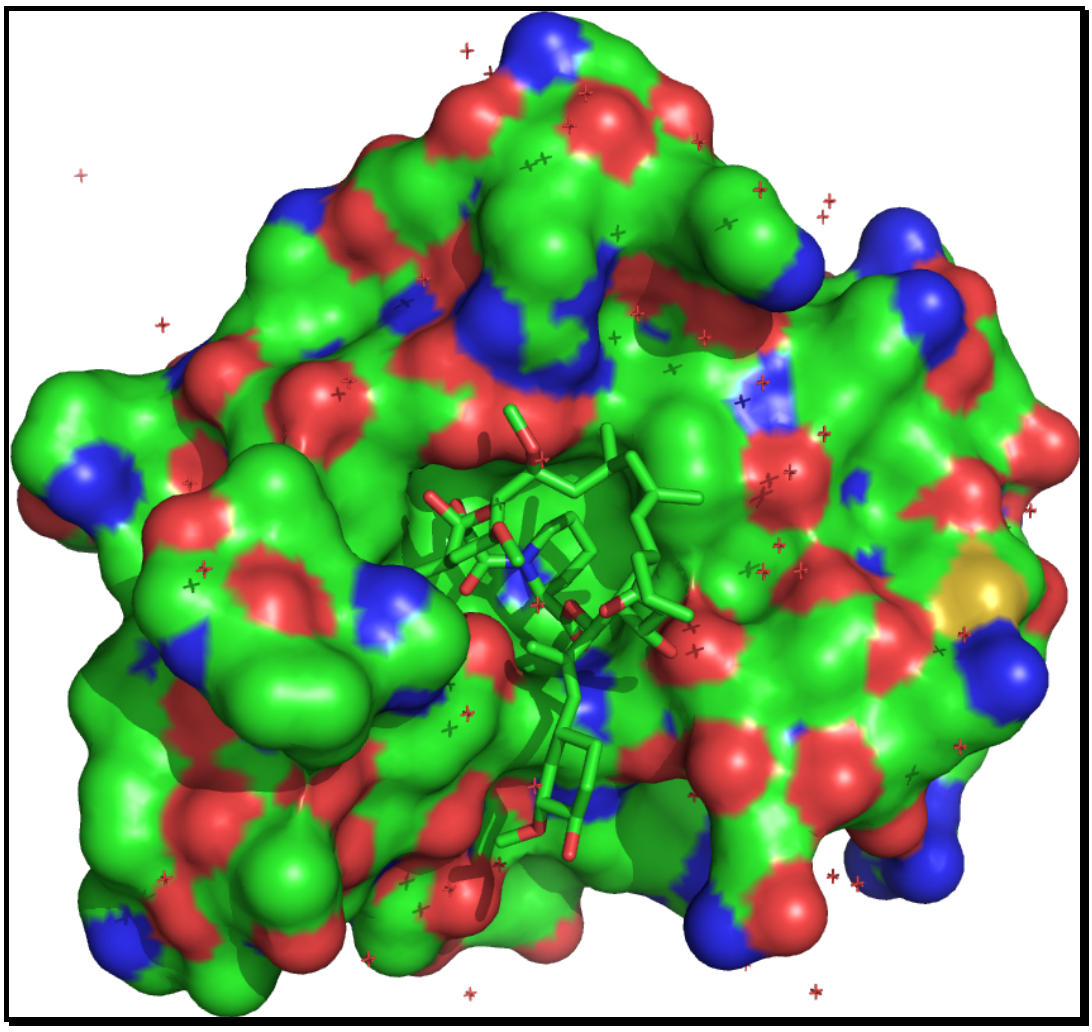


Figure 1.11 – FKBP12 (PDBID 1FKJ). Protein shown with a Connolly surface, complexed with FK506 (shown in stick representation). Crystalographic waters shown as red crosses marking the positions of oxygen atoms.

2DG3 shows FKBP12 complexed with rapamycin and is the result of published work carried out by Fulton [40]. The complex is shown in

Figure 1.12 with a Connolly surface applied to FKBP12 and a stick representation of rapamycin completing the complex.

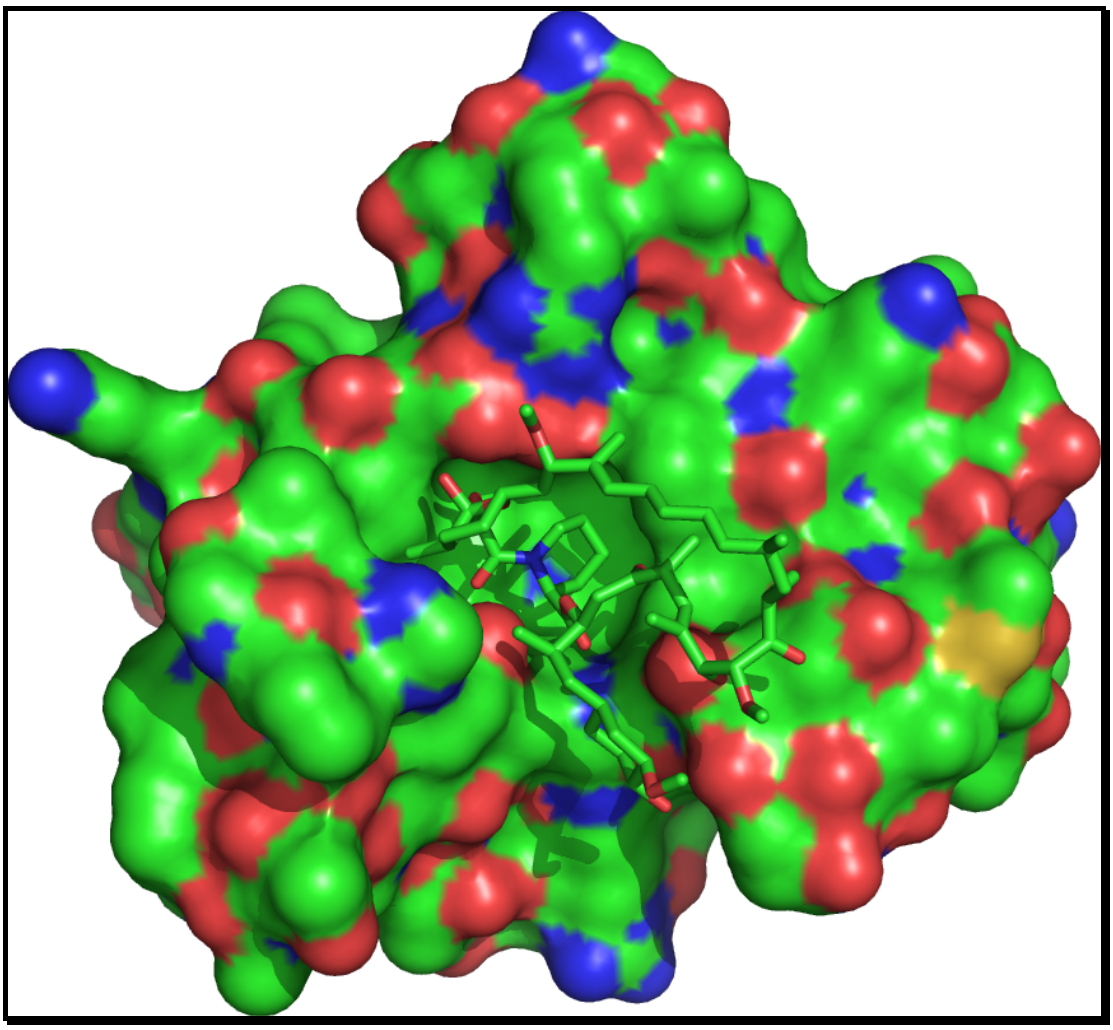


Figure 1.12 - FKBP12 (PDBID 2DG3). Protein shown with a Connolly surface, complexed with rapamycin (shown in stick representation).

NMR studies of small molecule binding to FKBP12 such as the one carried out by Stebbins [41] in 2007 helped build a picture of the active site and identify key interactions. Figure 1.13 shows the conserved nature of interactions being made with tyrosine 82 and isoleucine 56.

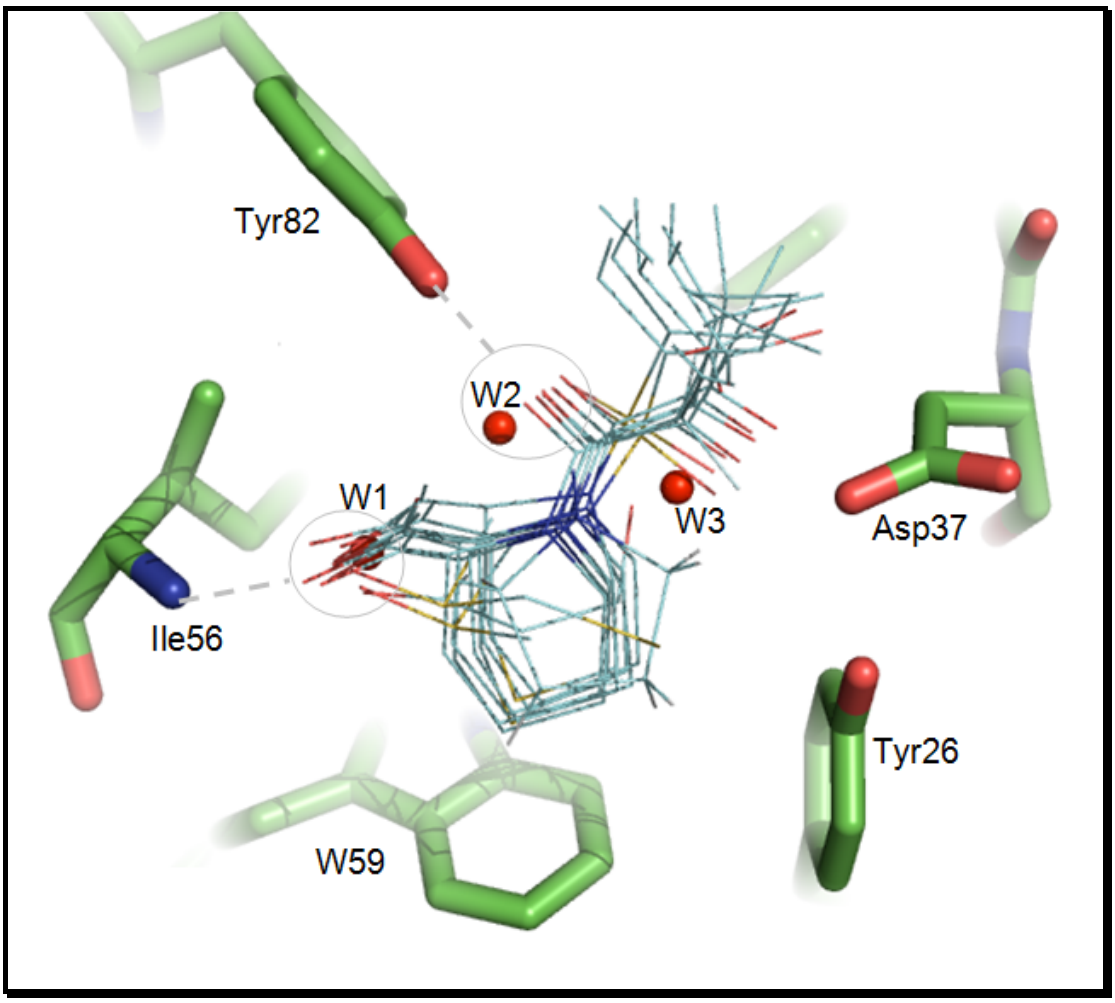


Figure 1.13 – The active site of FKBP12. Protein residues in stick representation, conserved waters labelled as "W" and an overlay of small molecule binders given in the line representation. Picture reproduced from the PhD thesis of Liz Blackburn [42].

1.5 Report structure

This report looks at ligand discovery from two approaches, structure based virtual screening and ligand based virtual screening; sometimes referred to as similarity searching. The introduction to virtual screening given here in chapter 1 gives a high level overview of the goals of virtual screening. Further background information on both structure based and ligand based virtual screening is given in chapters 4 and 5 respectively. Chapter 2 introduces some key concepts and algorithms which are made use of in later chapters. Chapter 3 deals with computer representation of biochemical entities. Chapter 6 concludes the work carried out, giving an analysis of the different techniques and their relative strengths and weaknesses.

Graphical representations of proteins have been generated primarily with the PyMol [43] molecular graphics program. 2D representations of molecules were created using the web applet Marvin [44].

2 High performance computing in ligand discovery: computer architecture and techniques

This chapter contains an overview of modern computer architectures, discussion of techniques for high performance computing including parallel computing, and algorithms used in the development of tools used during the course of this report.

2.1 Modern computing

In the last 20 years or so, modern computing architecture has remained almost constant. Distinct units have been defined which together make up a computing platform. The most important aspect of a modern computer is the Central Processing Unit (CPU). The CPU is the chip upon which instructions to manipulate data are carried out. CPUs are typically sold with a speed rating measured in Hertz referred to as the clock speed. This clock speed can be thought of as a timing signal, and upon each beat, it is generally the case that an operation is carried out. At the time of writing, it is common to find CPUs clocked in low multiples of Gigahertz (GHz) for example, the Intel® i7 CPU is clocked at speeds of up to 3.2 GHz [45]. In the early days of integrated circuits, when clock speeds were in the low multiples of Hertz, it would hold true that a chip running at twice the clock speed of another would execute a program in twice the speed. This is no longer true. As manufacturing techniques improved and CPU clock speeds increased at an almost exponential rate, other components, which are crucial in supplying data to the CPU had their speed increased, but certainly not in-line with the acceleration of CPU performance. Along with CPU clock speed increases driven by smaller components, the complexity and number of transistors within a CPU also increased. The trend is described and expressed by Moore's Law.

2.1.1 Moore's Law

In 1965, when integrated circuits were in their infancy, a short publication by Gordon E. Moore introduced the scope of the technology:

> *"Integrated circuits will lead to such wonders as home computers – or at least terminals connected to a central computer – automatic controls for automobiles, and personal portable communications equipment. The electronic wristwatch needs only a display to be feasible today"* [46; 47]

From this publication a general trend observed by Moore has become known as Moore's Law. The original text states:

> "*The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not increase"*.
>
> 46; 47

Whilst the publication deals primarily with the cost of putting components onto a chip, with the best economies of scale changing each year, this has been crafted into the generalised popular version stating that the computing power of microprocessors doubles every 18 months. The differences in time frames (doubling every year to 18

months) is explained in an interview with Moore in which he states that his original prediction was too short a time period, and quickly the trend established as doing slightly better than doubling the number of components every two years. His statement was changed by a colleague at Intel, the well known microprocessor manufacturer, and eventually used by the company stating that the processing power of microprocessors will double every 18 months [48]. The law has remained roughly accurate even now, over 40 years after its original conception.

2.1.2 Modern computer architecture

Within the CPU itself, there are many distinct units; one of the most important being the Floating Point Unit (FPU) which carries out mathematical operations. The FPU can be thought of as the calculator belonging to the CPU and is not concerned with logic operations such as controlling the flow of a program or dealing with memory; it only deals with arithmetical operations. When supplied information, a calculation is carried out and the result passed back to the CPU hardware for further logic and manipulation. An FPU may be able to carry out two arithmetical operations per clock cycle, effectively performing calculations at double the clock speed.

When developing programs for modern computing, it is important to understand the latency hierarchy that is present. Latency and its difference to bandwidth can be illustrated with a simple example. Using a Morse code key to type out a book to someone could be considered low latency (the individual data items are transmitted

almost instantly) and low bandwidth (it would take a long time to send a whole book using Morse code). An alternative high latency and high bandwidth approach would be hand delivering the book. To start receiving the data takes a long time (high latency) but once the handover starts, the bandwidth is high (giving the other person a book). Whilst the FPU may be the fastest piece of hardware in a computer, as the layers surrounding it are stripped away, the operating speed and bandwidth available decrease. With the CPU effectively running at half the speed of the FPU, full FPU performance can still be achieved through the use of pipelines. When instructions or code enters a CPU, logic can be applied and out of order execution techniques brought into use. Calculations that do not depend on each other can be performed in parallel. Parts of the program which can progress independently of each other can be put into separate pipelines. These pipelines act as a queue of instructions waiting to be executed. This chip level parallel execution may allow the FPU to achieve its peak performance of two arithmetic operations per clock tick. With the FPU and the CPU operating at massively fast speed, the problem of keeping them busy comes into play. With each step away from the CPU, data bandwidth decreases and latency increases.

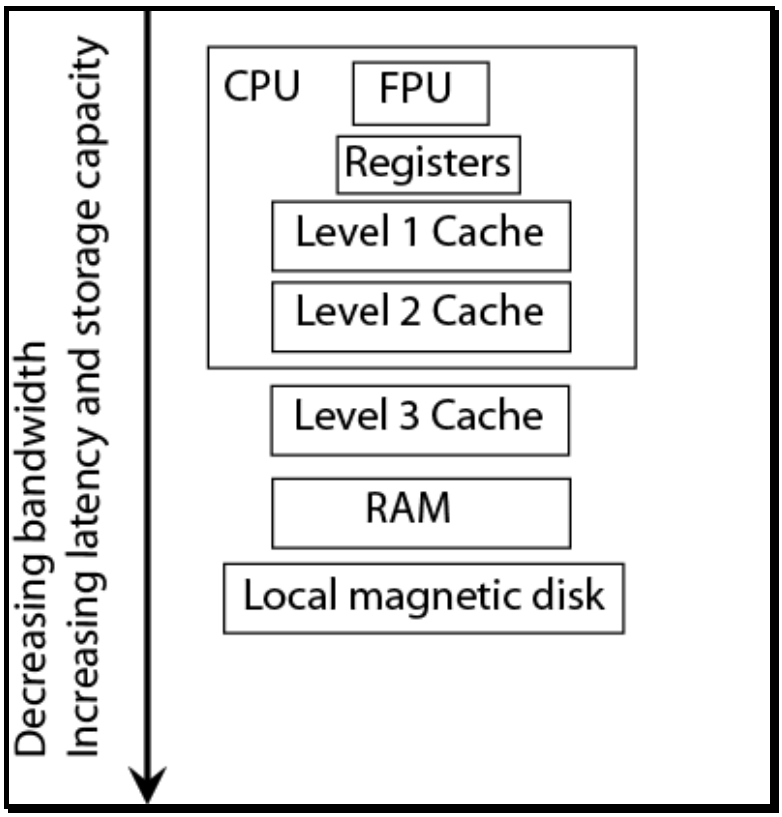


Figure 2.1 - Overview of modern computer architecture, flowing downwards, bandwidth decreases, latency and capacity increase

Figure 2.1 above shows the hierarchy of components a program must traverse to go from a local disk to finally calculations being carried out in the FPU. During the discussion below, the use of terms relating to speed such as slower refers to an increase in latency (time it takes for the component to access the data) and a decrease in bandwidth (the total amount of information that can be put through the component in a given time period). A program may well be stored on magnetic disk; this is the slowest and furthest away point from the FPU that will be considered. Upon program execution, this piece of data may be read into local RAM. From RAM it can then (if the hardware exists) be copied to level 3 cache. Cache memory is a small amount of very fast memory, many times quicker than RAM. Cache memory

aims to speed up the access time required for the CPU to address data in RAM. Once a piece of data is stored in cache, it can be accessed from this fast medium, and main memory or RAM need not be addressed. There exist many levels of cache memory; modern computing architectures usually employing three levels. Level 3 cache is located on the motherboard or main board of a computer, just off of the CPU. Usually within the CPU exists level 2 and level 1 cache, level 2 being larger but slower than level 1. The use of this cache memory enables the CPU to access data at lower latencies and higher bandwidths than if it was stored in the larger RAM. Keeping CPUs and FPUs busy is mostly facilitated by these caches, keeping data needed for calculation as close as possible to the CPU and FPU. A step faster but smaller in capacity than cache memory is register memory. Data being operated on by the CPU will be handled in CPU registers, which are clocked at the same speed as the CPU but where there exists only enough space for a few items of data. Techniques discussed later regarding program execution deal with keeping the CPU busy with limited communications channels.

2.1.3 Flynn's taxonomy of computer architecture

Flynn's taxonomy [49; 50] is a classification system for computer architectures which identifies two types of data and instruction delivery/operation methods [51]. Using Flynn's taxonomy, four different classifications exist. These are:

- single instruction, single data (SISD)
- multiple instruction, single data (MISD)
- single instruction, multiple data (SIMD)
- multiple instruction, multiple data (MIMD)

Figure 2.2 shows these architectures along with conceptual data flows and components.

The simplest machine is of type SISD, where a single processor executes its own list of instructions upon a single source of data. The least prevalent classification within the taxonomy is MISD, multiple instructions being carried out on a single data source. This architecture is present more for reasons of completeness than to represent a significant proportion of computer architectures. The SIMD architecture has in the past been greatly represented by vector machines. Large contiguous pieces of data (a vector) have the same operation performed on every element. Whilst this architecture is in decline, it is still seen in specialist fields; a vector processing unit was added to the UK's HECToR (High-End Computing Terascale Resource) in August 2008 [52]. The relatively new HECToR service (installed in October 2007) serves as a supercomputing resource for UK academics, the vector processing units added to it assist in the performance of codes suitable to vector computation. MIMD is the most prevalent architecture today, where computers made up of multiple processing units can execute their own instructions on their own data. They can be thought of as distinct computers tied together through some form of communications channel.

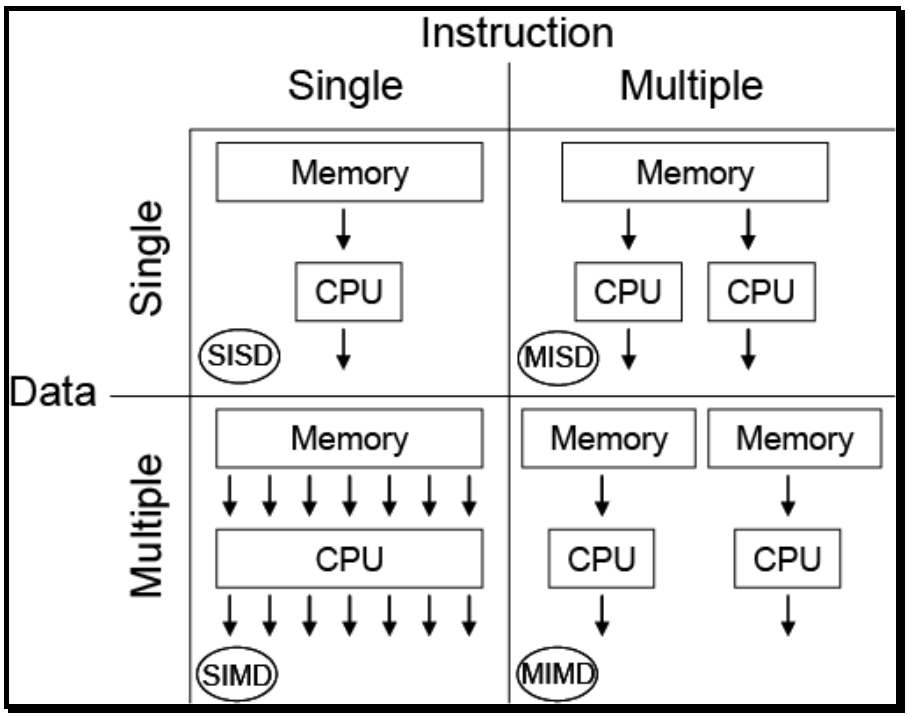


Figure 2.2 - Flynn's Taxonomy of Computer Architecture. Arrows show general movement and source of data when being accessed by the CPU

2.2 High performance computing

This section will give an overview of parallel computing and also the encompassing term high performance computing (HPC). High performance computing defines any computing effort that involves large amounts of computation. This can apply to running codes on parallel computers ('supercomputers'), or the use of extremely efficient codes on traditional computing platforms. The main point is that a high performance computer program uses available resources in a way that a simple everyday program does not. Outside of computer games and other 3D applications, little care is taken in the development of programs to ensure that they make optimum use of the hardware that they are running on. A word processor does not have to be highly optimised for speed and CPU time to be usable in everyday situations. The

golden disciplines for high performance computing are science and computer games. Using computers to model anything remotely complex from the real world requires vast amounts of computation. Increased computation leads to longer runtimes which may become so excessive that the investigation is not feasible. High performance computing essentially deals with a massive amount of data and/or a massive amount of computation.

2.2.1 Programming languages

Prior to discussing programming languages, it is important to note that no matter what language a program is written in, the computer is only able to understand machine code specific of the target platform. With this in mind, programming languages are essentially tools to aid in human understanding and allow programs to be written in a less strict and less complex way than pure machine code. This expression of the program or computation to be carried out is then translated by the compiler into machine code, often going through an intermediary step of assembly language. Figure 2.3 shows the stages a program written in a high level language goes through before being run on the target hardware.

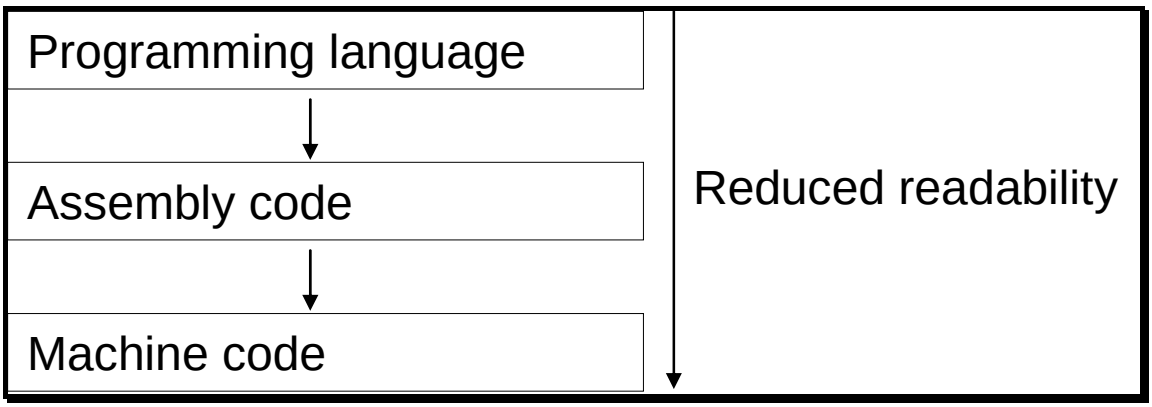


Figure 2.3 - Compiling a program often results in assembly language code being generated before translation into machine code

Assembly language is a human readable form of machine code, with each statement mapping almost directly to a machine instruction. Whilst assembly language is human readable, the complexity of writing programs in assembly language is not far removed from machine code binary. With literally hundreds of programming languages defined and often multiple compilers available for each and every target platform, the choice of language to use (and compiler to some degree) is crucial.

This section will examine the big three languages in scientific computing. These are FORTRAN, C/C++ and Java.

FORTRAN

Named from the IBM Mathematical Formula Translating System created in 1954, this was the first programming language which abstracted complexity away from the

programmer. Before FORTRAN, coding was carried out in assembly language, since assembly language mapped almost directly to machine instructions. The explicit nature of the statements made programming difficult. Historically, FORTRAN is the programming language used for scientific computing [53]. The extent to which compilers have been optimised ensures that generated machine code is often near optimum and little or no advantage would be gained over developing the program in assembly language.

C/C++

Although technically two languages, with C++ being an extension of C, the syntax and keywords are mostly the same, with all C programs being valid C++ programs. C began in the early 1970s created by D. M. Ritchie whilst working at Bell Labs USA, most of the language being implemented by 1973 [54]. Working in the same laboratory, C++ was later developed by B. Stroustrup in 1979 [55]. Originally named "C with classes", C++ offered a more human readable method of abstracting data structures and ideas.

Java

Developed by Sun Microsystems in 1990, Java aimed to address some limitations present in the traditional compilation model of programming language translated to assembly language, then translated to machine code. The main difference with Java

is that a program is converted into Java byte code instead of machine code. This byte code is machine code for a computer which is emulated in software. This allows compiled Java code (byte code) to be portable across a range of platforms. In essence, a compiled Java program contains instructions for a non-existent machine which is emulated in software. This gives Java the property of being run on any hardware platform for which the Java virtual machine (which emulates a computer understanding java byte code) is available. Using this approach, emulating a hardware platform is not as efficient as running code natively due to many overheads. To address this issue, a technology known as "Just in time" (JIT) compilation was developed. Essentially, the code is run through the Java compiler to create a Java byte code file. When the program is to be run, it is given to the virtual machine. The virtual machine can then run the code in an emulated fashion or use JIT compiler technology. JIT looks at the Java byte code and translates instructions into machine code native to the platform on which it is running. This JIT compilation step is paid for with time taken for JIT code to be created and can often lead to slow start up times.

A comparison of languages

Choosing a programming language for scientific computing is a subject of great contention. Whilst Java and C/C++ are the most easily human readable and FORTRAN the least, an obvious choice would be to go with Java or C++. Historically, Java would have been ruled out due to the emulated nature of its execution; however, JIT compilation technology has closed the Java/C/C++

performance gap. Whilst benchmarks can give an indication as to the efficiency of a language and the technology underpinning it, a language may produce code that performs well in one benchmark and poorly in another. Putting these discrepancies aside, it is held that FORTRAN produces the most efficient machine code, followed by C/C++ and finally Java produces the slowest [56].

One study of the three languages uses matrix multiplication to benchmark performance. The metric measured is MFLOPS (Million Floating Point operations Per Second) of Java, C/C++, FORTRAN and ESSL (IBM Engineering and Scientific Subroutine Library) [57]. ESSL is a hand tuned set of function calls that can be integrated into other programming languages, and provides highly optimised routines for specific computational operations. In the study by Moreira [53], ESSL routines were called from a FORTRAN program. Figure 2.4 shows the performance in MFLOPS achieved by each code and a simple matrix multiplication program.

Language	Matrix size	MFLOPS
Java	64x64	2.2
Java	500x500	1.6
C/C++	64x64	137
C/C++	500x500	91.1
FORTRAN	64x64	205.4
FORTRAN	500x500	193.3
ESSL	64x64	253.2
ESSL	500x500	248.3

Figure 2.4 - Benchmarking matrix-matrix multiplication codes in Java, C/C++, FORTRAN and ESSL (with FORTRAN). Performance measured in MFLOPS (higher is better) [53]

Whilst the performance of Java reported in Figure 2.4 is low compared to the other languages, it could be suggested that Java is not well suited to this benchmark. Additionaly Java performs checks not carried out by the other languages on the matrix, such as looking at the value and deciding if it is valid and computation can continue, rather than trusting that the programmer has initialised the matrix with valid values and continuing computation. Turning these checks off reportedly results in the maximum performance of 2.2 MFLOPS becoming 33.3 MFLOPS [53]. The performance gap between Java and C++ is shown to be closing in a study by Bull [56] as compiler technology advances.

2.2.2 High performance computing techniques

The following section will look at the engineered example of running a molecular dynamics program to simulate a protein in solution. This may be done using many of the established and available molecular dynamics programs such as GROMACS [58; 59; 60] or NAMD [61] but we will assume that the code to carry out the computation is custom-written and runs on a single processor. This code may take in the order of days to calculate a time period within the simulation. Standard HPC techniques may be applied to the code in an attempt to reduce the computation time; Figure 2.5 shows the many ways in which this can be achieved and the associated effectiveness and difficulties.

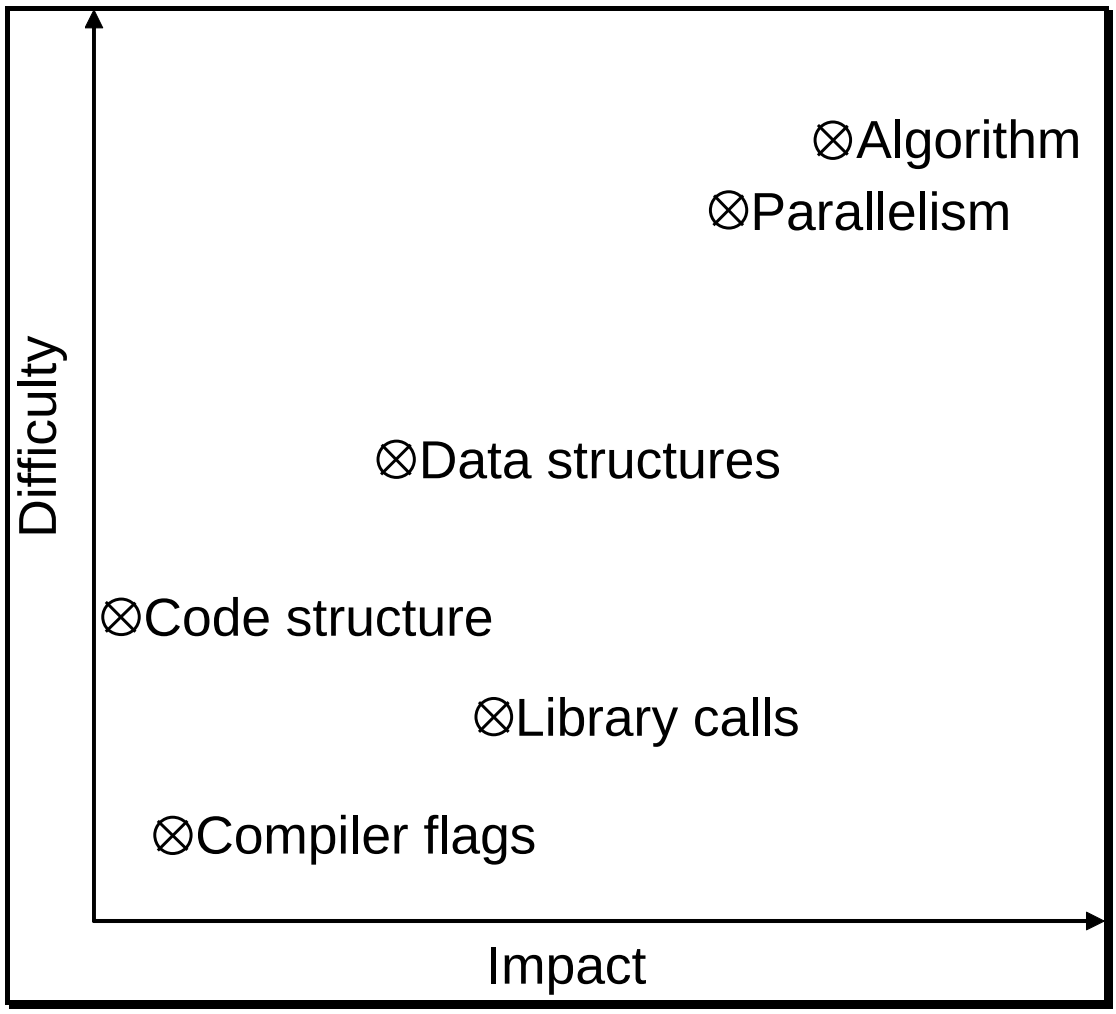


Figure 2.5 - Optimisation techniques arranged by impact and difficulty

Applying the techniques shown in Figure 2.5 enables high performance computing.

2.2.2.1 Compiler flags

The easiest way to attempt program optimisation is to instruct the compiler used to produce the final executable version to attempt optimisations. Depending on the code and the way in which it is written, compiler flags can have a varying degree of effectiveness. Simple compiler flags, such as specifying the exact microprocessor that the code will be run on may allow the compiler to make use of features of the chip otherwise left out of others. The most commonly used compiler switches describe the level of optimisation that should be attempted. For example, the GNU Compiler Collection (GCC) [62] allows compilation to take place guided by a number of compiler switches used to specify optimisations that the compiler should attempt. The most common way of implementing compiler optimisations using GCC is to use the switches -O, -O1, -O2 and -O3. These switches are really shortcuts and combine sets of singularly selectable options into convenient lists specifying the level of optimisation required. Whilst the success of optimisations depends heavily on the code being compiled, the general idea is that a program compiled with –O3 will have more optimisations attempted upon it than a code compiled with the switch -O1. As the amount of effort required to turn on compiler optimisation is minimal, it is often practice to compile code at the -O3 level of optimisation. However, situations arise when this is impractical. For example, using -O3 may produce larger executable sizes than code compiled at level -O1. The code may be of a larger size due to many factors, such as loop unrolling (detailed later) and constants used in place of runtime generated variables. Although the produced binary is larger in size, it is hoped that the execution time would be reduced in this new format. The compiler switch -O1

instructs the compiler to attempt a number of optimisations, a few interesting ones are detailed below.

- -fguess-branch-probability. Conditional statements of the form “if (condition) then do this, else do that” are expensive for modern microprocessors; expensive in the amount of time taken to deal with the statement. Most modern microprocessors are built with a number of instruction pipelines. These pipelines are very fast but small pieces of memory which act as a queue of instructions that the chip should execute. Keeping these pipelines full is a major challenge to both programmers in the way their programs are written, and for developers of compilers who should ensure that binaries produced make the best possible use of the available pipelines. When a program is executed, it is loaded into memory that is addressable by the microprocessor. If we consider code being loaded into the RAM of a typical desktop computer, we see that the memory is running at a much slower speed than the CPU; perhaps 400 megahertz as opposed to 2 gigahertz. Moving data from memory to CPU takes time, and if the CPU is not busy performing other operations, loading this data causes the chip to wait or ‘stall’. It is therefore advantageous to keep the processors busy with their pipelines full. When a conditional statement (such as “do this if A greater than B, otherwise do this”) is reached, which may cause branching within the code, the processor may find itself in a situation where the branch has taken program execution to a piece of code not positioned next in the

pipeline. Reordering or reading from slower memory may be required and cause a pipeline stall. The switch -fguess-branch-probability causes the compiler to use heuristics to attempt to predict the outcome of conditionals and provide instructions in an order that are most likely to minimise pipeline stalls. In a simple example, where there is a branch on the condition that variable 'A' has a greater value than variable 'B' and this is true 99% of the time, then the compiler will reorder code to exploit the frequency of this condition being satisfied. Branch probability can be exploited in the example of molecular dynamics code in many trivial ways, such as the testing applied to see if the required amount of simulation time has been achieved. In all but one case (when the final iteration has been completed), the code should continue simulating time steps within the system.

- -floop-optimize. Most program execution time is spent inside loops. Often scientific codes such as those calculating forces between particles execute the code inside loops billions of times. Amongst other operations, this option examines loops for unchanging and constant expressions between loop cycles. Removing these and performing them only once before entering the loop can remove a lot of unnecessary code execution. Loops can be controlled in many ways using compiler switches, and techniques such as loop unrolling can provide large performance increases in many situations. Take for example a loop within a loop. The deepest loop is situated over the second dimension of an array of floats (numbers represented in the IEEE 754

standard, discussed later). As this second dimension represents the three Cartesian coordinates in the molecular dynamics example required to represent a point (or atom) in space, the innermost loop can be replaced with three direct statements operating on the correct data. One loop has been removed and only the outer loop remains. As there are overheads associated with loops such as the testing of exit conditions, this overhead has been removed. The programmer would therefore expect to see a decrease in execution time for their code.

A simple loop unrolling example is shown in Figure 2.6. The example code shows a nested loop in the upper portion of the figure. The loop has been unrolled in the lower portion. As it is known that the loop is over the x, y and z coordinates of the atom, the loop can be removed and the statements coded directly. If the operation was on thousands of atoms, then thousands of loop initialisations and check on exit conditions (when to stop looping) have been removed. A loop unrolling example is given in Figure 2.6.

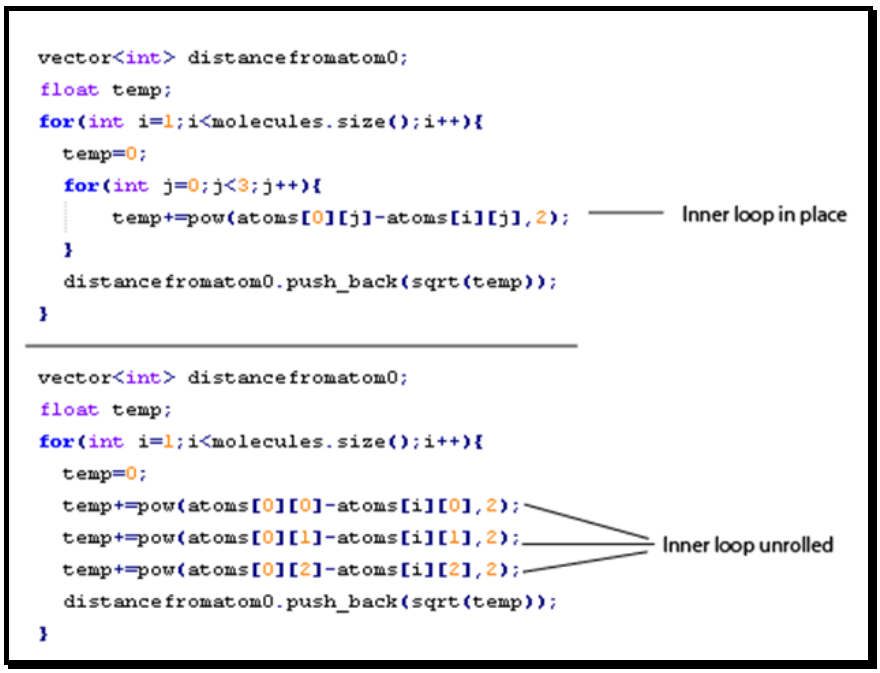


Figure 2.6 - Loop unrolling example. Two equivalent code examples, the lower with an inner loop unrolled showing the removal of one loop and therefore reducing loop overheads encountered at each iteration of the outer loop.

The effectiveness of compiler flags depends both on the language being compiled and the quality of the compiler being used. Often certain languages are better suited to compiler optimisation; a popular feature of both the C and C++ programming languages is the ability to use pointers and directly manipulate memory. This is problematic for compilers aiming to make executables for modern CPUs which use pipelines and out of order execution. Upon compiling a program which makes use of pointers, the programmer is effectively managing the memory. This means that the compiler cannot see when a memory location may be altered or read from. A program that does not make use of pointers has all of its memory requirements mapped to variables with reads and updates visible to the compiler. With the access patterns known in the program that does not make use of pointers, the compiler can

employ out of order execution and heavily pipeline the program, knowing when memory locations will be accessed, and ensuring execution proceeds in a manner equivalent to running the program in a serial manner on one pipeline, the goal of this pipelining ultimately being a speed increase.

2.2.2.2 Library calls

The phrase “reinventing the wheel” is often heard in software development referring to putting effort into solving a problem yourself for which there already exists a perfectly valid and available solution. Many scientific codes rely on sets of already well defined mathematical techniques, most of which will have been implemented in the past by a programmer and are available for use. Performing matrix-matrix multiplication is a fairly trivial problem to implement a solution to, requiring only a small amount of code. However, the implementation may not be stable, correct or fast for all cases. In this situation, the programmer should avoid their own implementation and use a suitable library. The Basic Linear Algebra Subprograms (BLAS) [63] library was created for the purpose of operations on scalars, vectors and matrices. Using appropriate libraries ensures that the implementation is both correct and fast. Microprocessor vendors also release versions of libraries such as BLAS which are highly optimised for their chips. Another example of a library is “Fastest Fourier Transform in the West” (FFTW) [64]. This library implements Fourier transforms in an arbitrary number of dimensions. The code can be downloaded and a library built with optimisations specific to the target platform. Library calls are easy to implement if the correct library is found and a good level of documentation is

available. Simply replacing code or calls to functions with library calls can greatly improve performance. Additionally, under extreme conditions (e.g. when the data supplied makes the operation impossible), it will insure that the code performs predictably. In the example of the custom written molecular dynamics code, vector operations performed on the coordinates of atoms and their velocities can be carried out using calls to the BLAS library.

2.2.2.3 Code Structure

Code structure can affect execution time in a number of ways. Primarily, the order in which operations are carried out can affect execution time. Redundant expressions in loops which are unnecessarily recalculated can also be removed. Short functions that are called many times can have their calling statements replaced by the actual code in the body of the function, removing the need for a function call. Whilst compiler optimisations turned on with switches attempt to perform these rearrangements, code may be too complex to be operated upon. Changing the code structure may simplify things allowing the compiler to do a better job of optimisation.

2.2.2.4 Data structures

Data structures used in programs can be thought of as mappings to memory locations. The layout that these structures use can often not be ideal. For example, if a program represents the position of particles as a 2D array, the first dimension being the particle number, and the second dimension containing three entries representing the x, y and z coordinates of the atoms in Cartesian space, how this maps to memory is dependant on the programming language used. The C and C++ programming languages lay out array in memory using column order [65; 66]. This means that the data in index 1,2 is next to 1,3 in memory as shown in Figure 2.7.

Figure 2.7 - Array ordering in C and C++

The FORTRAN programming language maps array data to memory using row order [67; 68], the first index moving first. This creates the mapping shown in Figure 2.8.

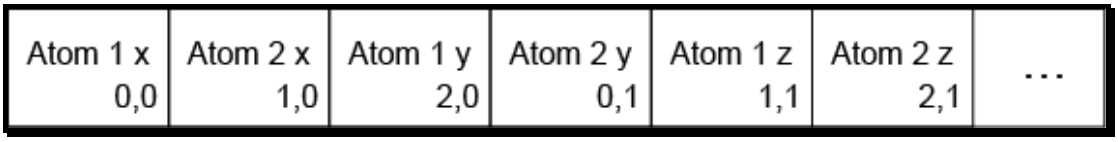


Figure 2.8 - Array ordering in FORTRAN

It may be of little concern how the compiler of the programming language used maps array data to memory for the average programmer. However, it is of importance when optimising code. For elements that will be read and used in the same calculation, it is advantageous for them to be stored next to each other in memory. So, if calculating the distance between atoms, the first step is finding the square of the differences in each coordinate between pairs of atoms. If the x coordinates are stored adjacent to each other in memory, then large read leaps around memory are not required and often data will be read into the processor in chunks [67]. From this simple example, the FORTRAN like row ordering is advantageous however, stating that the FORTRAN ordering style is advantageous in this example does not devalue

C and C++ ordering style as proper declaration of the array size, and careful value initialisation can produce the same ordering as shown in Figure 2.9. Awareness of data layout in memory and proper data management has the potential to reduce the execution time of the example molecular dynamics code dramatically.

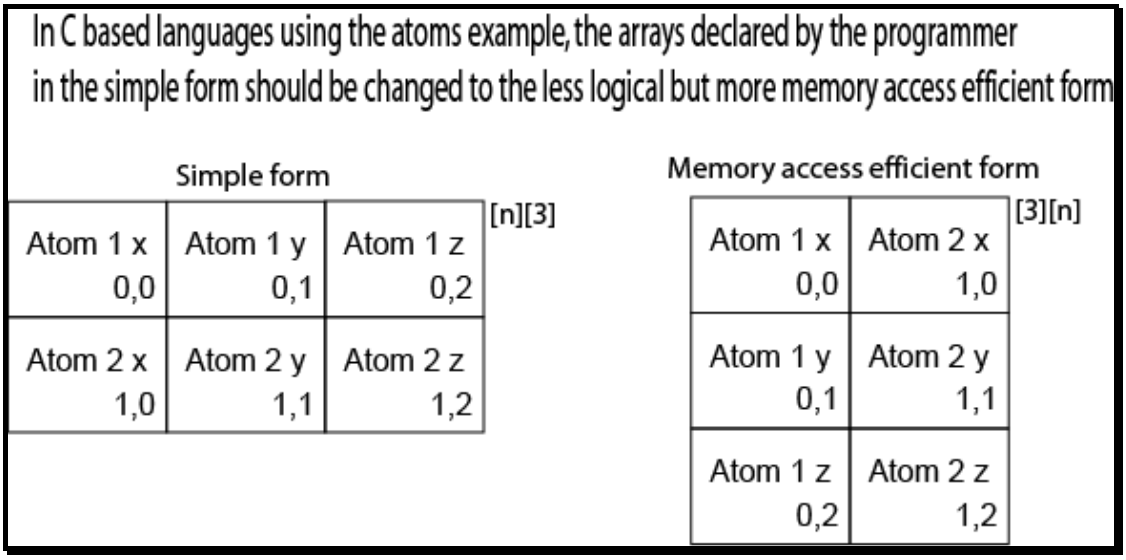


Figure 2.9 - Efficient memory mapping for atomic distance calculation in C based languages

2.2.2.5 Algorithms

Changing the algorithm used by a code to simulate real world situations or other mathematical problems is the optimisation that is most difficult and time consuming. This is essentially rewriting the main computational part of the code to carry out the computation in a different and hopefully faster and more efficient way. This may be fairly simple in a code for which the wrong choices for the algorithm have been made in the first place. In an overly simple example, the amount of rotation applied to an object represented in 3D to put it in a required orientation may be determined using an iterative minimisation routine. In this case, a slightly more mathematically

involved direct method could be used to find a rotation matrix which, when applied to the target object, produces the correct arrangement. This direct method would be quicker than the iterative method and provide exactly correct results while the iterative method may have halted once an answer was within a predefined tolerance, as is often the case. Using the correct algorithm for the task can provide more accurate results and cut the computation cost dramatically.

2.2.2.6 Parallelism

Most codes are suitable for parallelisation. However, the return on the massive effort required must be realised with either a significant reduction in runtime, the ability to perform the calculations on larger datasets or more complete results. The success and extent to which a code can be parallelised and the scale of the parallelisation is dictated by the data access patterns present, the algorithm used and the type of hardware that the parallel implementation will be run on.

2.3 Parallel computing

Using multiple processors for a problem is deemed “parallel processing”, as each processor will work alongside others in achieving a common goal. However, parallel processing cannot be put to use on all problems. The problem must be suitable for parallelisation. The suitability of a problem to be parallelised relies heavily on the choice of algorithm and code used to solve it. Often, an unsuitable existing code that solves a problem can be altered and changes made to the algorithm to allow parallelisation.

2.3.1 The argument for parallelism

The increasing computational ability of microprocessors will not continue indefinitely without a change in approach. There is ultimately a limit to the speed of which components can operate at.

As component sizes decrease, the fundamental laws of physics come into play. Generally, the more components added to a chip, the greater the amount of energy is wasted in the form of heat. There is obviously also a minimum size which components can be. In the far-fetched and extremely unlikely event that single atoms can be used as transistors, then this is a physical limit to the amount of components that can be placed on a chip. If the approach then taken is to just create larger chips to add more components, then the speed of light comes into play. Take

for example, a chip operating at a clock frequency of 2 gigahertz. For simplification, we shall say that the chip is 1cm by 1cm. For an electrical signal to travel diagonally from one corner of the chip to the other opposite corner, it must cover a distance of 1.41cm. As the speed of electrical transmission is governed by the speed of light, which we will take to be $3x10^{8}$m/s, we can see that the time taken for this signal to travel 1.41cm is $4.7x10^{-11}$ seconds. With a clock frequency of 2 gigahertz ($2x10^{9}$ cycles a second) an electrical signal can travel a distance of 15 cm in a clock cycle. Whilst the electrical signal can travel more than 10 times the distance required, it is plain to see that the limits of physics are being approached. With a higher clock speed of 10 gigahertz and a chip size of 3cm by 4cm, a signal travelling diagonally across the chip from one corner to an opposing corner (a distance of 5cm) would not complete the required distance, travelling only 3cm in a clock cycle (as shown in Figure 2.10). The path taken by the electrical signal is also not straight, being made many, if not hundreds of times longer by the internal logic pathways of the chip. This makes getting information in and out of the chip unachievable on each clock cycle. This example illustrates the general idea that the capacity of microprocessors is not infinite using well known techniques of today. We are reaching the end of Moore's law.

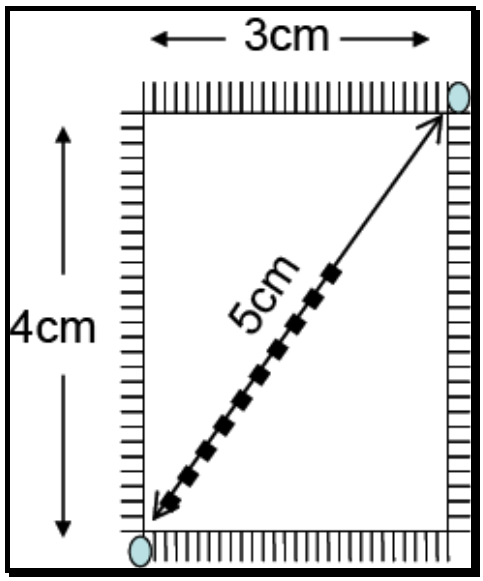


Figure 2.10 - An electrical signal travelling at the speed of light along a 5cm path, only travels a distance of 3cm as shown by the dotted line in the tick of a 10GHz clock

A way around this limitation imposed on single microprocessors is to use multiple processors when suitable. The use of multiple processors to address a problem or computation not only allows the theoretical barriers present today to be broken, but also allows for extracting and amalgamating the power of existing processors.

Gropp sums up the situation as follows:

> *"It is not only that the speed of light and the effectiveness of heat dissipation impose physical limits on the speed of a single computer (to pull a bigger wagon, it is easier to add more oxen than to grow a gigantic ox.) It is also that the cost of advanced single-processor computers increases more rapidly than their power (Large oxen are expensive.) And price/performance ratios*

become really favourable if the required computational resources can be found instead of purchased." [69].

2.3.2 Amdahl's Law

An important law in parallel computing is Amdahl's Law [70], which states that the speed increase obtained by running the code in parallel on multiple processors is limited by the serial portion. The rationale behind this is that any program will have a portion which must be carried out in serial (on one processor step by step). If this proportion was 10% of the entire runtime on one processor, then the remaining 90% can be parallelised and only from this a speed increase in computation achieved. Essentially, no matter how fast the parallelisable section of code is made, the time taken by the serial 10% remains. With enough processors reducing the parallel part runtime to negligible proportions, the runtime will be completely dominated by the serial portion being carried out on one processor. This observation at first seems to limit the effectiveness of parallel computing. However, whilst the serial portion cannot be improved through parallelisation, parallel techniques allow bigger problems to be tackled than would be attempted in a serial environment. This leads to the fact that larger problems are addressed and therefore the serial computation required becomes negligible. Amdahl's law would hold true and the effectiveness of parallel programming would be greatly reduced if problems of a serially manageable size were tackled; luckily, problems tackled by parallel computing are of a sufficient size to render Amdahl's law insignificant [71].

2.3.3 The fundamentals of parallel computing

The way in which a problem is parallelised depends heavily upon the algorithm used to solve it, the code and the computing platform that the parallel implementation will be run on. Taking these factors into account, the programmer or scientist undertaking the parallelisation task must understand the problem thoroughly as well as the way in which information flows during execution. They will then use judgement to parallelise the code using one or more (usually one due to complexity issues) parallel programming paradigms. Parallelism relies upon the fact that the calculation being carried out can be broken down into discrete chunks of work which can be carried out in a reasonably isolated and independent manner. If the entire workload cannot be undertaken in an isolated manner, then some form of communication between the work units must be made. This communication is an additional overhead to the code and if too much is needed, then communication may dominate. In parallel computing, communication is a major limiting factor. In an ideal world, taking a code that takes 16 minutes to finish execution on a single processor, parallelising the code and running it on two processors, the developer would like to see execution time cut in half to 8 minutes. This is rarely the case as parallelisation adds complexity, often requiring collaboration between the processors which adds to the runtime. This is a new level of complexity that did not exist in the serial code. In the previous example, of the code taking 16 minutes on one processor and 8 minutes on 2 processors, we can say that linear speedup has been achieved. In this case, the speedup takes the value of 2 on 2 processors. Speedup is expressed as a number ranging from greater than zero to infinity. Speedup is calculated as shown below.

$$S = \frac{T_1}{T_N}$$

Where S is speedup, T_1 is the time taken for the calculation on one processor and T_N is the time taken on N processors.

If a code ran for 16 minutes on one processor and can be run in 2 minutes on 8 processors, then the speedup factor is 8. This is linear speedup. Speedup can be expressed in another way called parallel efficiency and is expressed as a percentage. Linear speedup is equal to 100% parallel efficiency. Parallel efficiency is a good measure of how well a parallelisation technique is performing. If a parallel efficiency of 90% is achieved then the researcher knows that it is the equivalent of each processor performing the calculation at 90% of the speed of the serial implementation. Parallel efficiency is calculated as follows:

$$PE = \frac{T_1}{T_N N} = \frac{S_N}{N}$$

Where PE is parallel efficiency, T_1 is the time taken for the calculation on one processor; N is the number of processors the parallel code is being run on, T_N the time on N processors and S_N is the speedup on N processors.

An example which shows typical speedup and parallel efficiency is available in the book High Performance Computing, published by O'Reilly [72]. A simple numerical code written in the FORTRAN language is profiled on a 16 processor MISD

computer. Profiling the code on a range of processors produced the timings and calculations of speedup and parallel efficiency shown in Table 2.1.

Processors	Time	Speedup	Parallel Efficiency
1	30.9	1.0	100%
2	15.6	1.98	99%
4	8.2	3.77	94.3%
8	4.3	7.19	89.9%
12	14.2	2.18	18.2%
16	57.0	0.57	3.6%

Table 2.1- Example speedup and parallel efficiencies. Adapted from [72]

Table 2.1 shows typical scaling of a code which is not suited to a large number of processors.

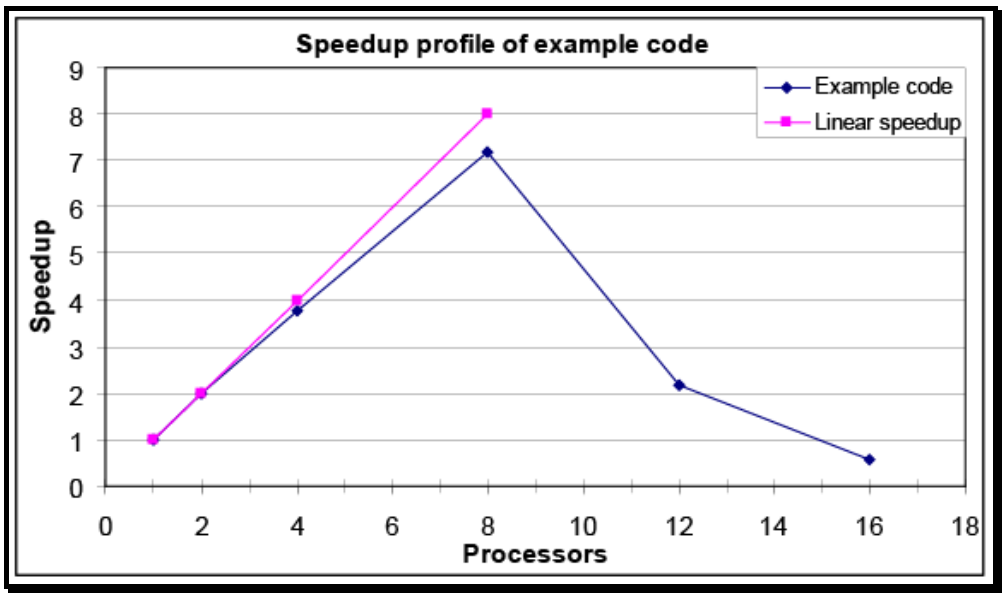


Figure 2.11 - Speedup profile of example code

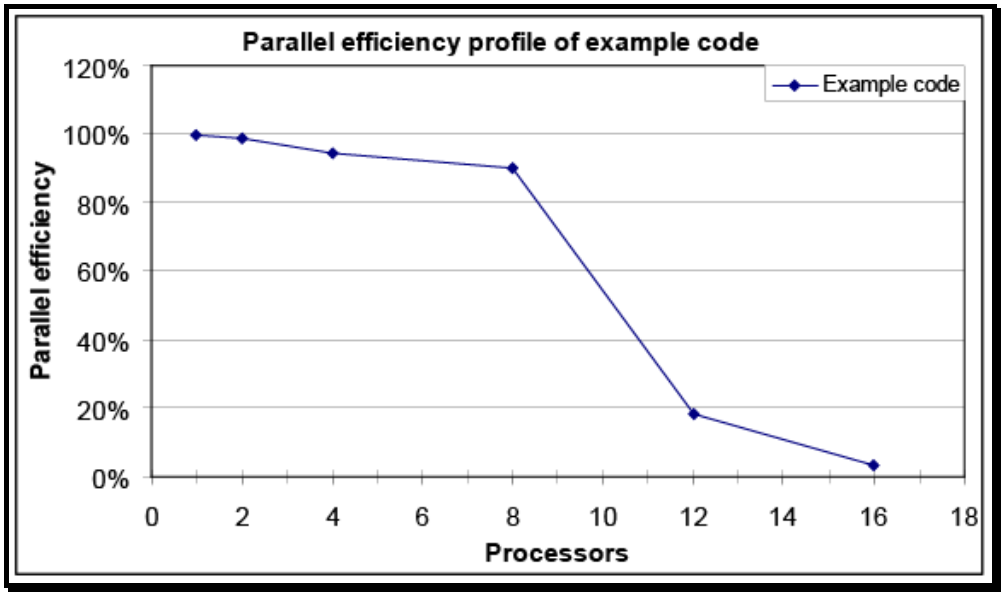


Figure 2.12 - Parallel efficiency profile of example code

Figure 2.11 and Figure 2.12 show respectively speed up and parallel efficiency of the example code across a number of processors. The characteristics are typical of a

code that benefits from a small scale parallelisation, but as the number of processors increases, the benefit rapidly decreases, becoming detrimental after eight processors. When parallelising a code, it is often challenging to get the code to scale to larger numbers of processors. This of course depends on the characteristics of the problem being tackled, the implementation choices made by the programmer and the overall amount of computation required. For example, if the code being parallelised involves molecular dynamics simulation of an aqueous volume decomposed over a range of processors, then at regular intervals, communication must occur in order for each processor to broadcast the new location of atoms that it is working with. It is often the case that the data access patterns are unsuitable for parallel computation across a large number of processors. As processors are added to the problem, many different effects come into play. As the example code is being run on MISD hardware, cache coherency becomes a problem as the number of processors used increases. Data that is stored in cache memory on one processor may cause a slow down in execution time; caches have to be coherent, changes must be populated back to other processors or RAM more frequently, adding overhead and additional operations for the hardware to perform.

2.3.4 Parallel programming paradigms

A scientist parallelising a scientific code may choose from a range of parallel programming paradigms to achieve the parallelisation. These paradigms include message passing, shared variable and data parallel techniques.

Message passing

This approach uses explicit messages sent between processors or execution threads to provide communication between work units. This is the most common way of implementing a parallel program and can be used on almost any hardware with messages being passed on custom interconnects or standard network hardware. The Message Passing Interface (MPI) has become the standard software library for implementation of message passing programs [69]. There exist many implementations of MPI, some vendor specific and others free and able to run on many platforms. Whilst MPI provides the ability to send and receive messages between processors, it is still up to the programmer to implement these messages and design a pattern for the processors to follow when exchanging data. The time or point of execution when messages are sent or listened for must be coordinated amongst processors. This creates a communication pattern. A well known message passing pattern is the task farm. A task farm is a suitable method of parallel decomposition when the work being carried out can be reduced into distinct chunks which require none or little communication between themselves. A task farm typically consists of a master process which farms out the work packets to worker processes. Upon execution of the program, one master process will send out work packets to waiting worker processes. The workers perform operations on their packets and then ask the master for more once finished. A great advantage of the task farm is its simplistic nature, and also the way in which (with enough small work packets) load balancing is easily achievable; for example, one worker will not receive many times more work than other workers. As time goes on and workers complete their tasks, they ask for more;

a worker with a troublesome task that requires a lot of computation will only request more work once computation on the current packet has finished. A worker with a massive amount of work when the others have none (load imbalance) causes the whole programs execution time to increase; even though many workers may have finished, the program is still running somewhere. It is obviously best to keep all workers as busy as possible and aim for all to finish at the same time. As mentioned before, this is one of the task farm's strong points, and when the problem being tackled is suitable, it is a strong candidate for the parallel decomposition and communication pattern.

Shared variable

Shared variable programming is an extremely fast method of communication amongst a small number of processors which share the same memory. These computational platforms are of the MISD architecture type and rely upon the fact that a commonly addressable memory is accessible by all processors [73]. As processors must share common memory, the task of managing caches becomes a problem. With caches being so important in achieving high performance and keeping data close to CPUs, they cannot be removed, and with them, the issue of cache coherency comes into play. Additional hardware is required to ensure that each CPU sees the same data when operating on a memory location mapped to a local processor cache. An update by one processor to a location which is not pushed out to main memory may mean that another processor sees an old version of the data. As mentioned, additional hardware is required to ensure coherency between the caches. For this

reason, the number of processors sharing a cache coherent common memory store is low, typically eight processors. Additional processors add to the complexity of coherency hardware, and so adding more processors becomes less advantageous. Shared memory programming techniques coupled with a MISD architecture machine are ideal for certain types of problem for example, operations on large matrix data and especially problems regarding images, where the common data can be operated upon by multiple processors at the same time but data being changed by one processor may be required by others at the same time. MISD hardware also provides hardware to ensure that race conditions do not pose a problem; two processors accessing the same memory location simultaneously may cause problems. This is called a race condition and logic exists to ensure that a processor does not read from a location when another processor is altering or using in a destructive way the data stored within it. The hardware has the ability to apply a lock to a location or ensure that access to it is carried out in an atomic manner. The standard library and method of implementing the shared variable paradigm is OpenMP [74].

Due to the relatively small number of processors that can share a commonly addressable memory, often many small groups of processors can be tied together with some message passing hardware acting as an interconnect. This is called mixed mode programming. Mixed mode programming is a mixture of shared memory parallelism between groups of CPUs sharing the same memory, and, between these groups, an ability to pass messages exists. This gets around the problem of the rising complexity of cache coherency with increasing numbers of processors on MISD

architectures. The cost however, is that the type of problem easily addressed may become more specific and the complexity of the parallel implementation increases greatly.

It is possible to use the shared variable paradigm on non MISD hardware, such as MIMD devices [75]. However, this only abstracts away the fact that messages are being passed and allows programmers to write code as if it was being executed on MISD hardware. This extra layer of abstraction, and the programmer not being able to directly control messages inevitably removes the ability to write highly optimised code.

Data parallel

Data parallel is a technique used when development speed is required. Often, the code produced is not as fast or well suited to parallel computing as code developed using what is seen as a lower level and more powerful parallelisation technique such as message passing or shared variable [76]. The parallelism is expressed by distributing the data to be worked on over a number of processors. This technique relies heavily on the compiler having explicit knowledge of the platform upon which the code will be run on, and guided to some degree by the programmer with the addition of keywords to the source code. Parallel execution is usually achieved in the same way as used in shared memory techniques, decomposing loops. Data parallel techniques are seen as a higher level than shared memory [76] or message

passing techniques, essentially achieving the same goal as the shared memory paradigm but being able to achieve this using both this technique and message passing which may be required if the target platform does not have all processors using the same addressable memory. Data parallel techniques could be seen to offer less control than the previous techniques and also the portability (movement between different hardware configurations) of the code is greatly reduced, implementations in the previous paradigms offering a wider variety of suitable hardware for execution.

2.4 Numerical stability and its relevance to scientific computing

With computing well integrated into our everyday lives, people are often astonished at the inaccuracy present in certain types of everyday computer arithmetic that is carried out. These inaccuracies, although small, can be amplified many times due to the repetitive nature of the work carried out on microprocessors.

Put simply, $(a+b)+c$ is not equal to $(c+b)+a$.

Two examples showing the failures of floating point arithmetic on microprocessors are shown in Section 2.4.2 Herron's formula and Section 2.4.3 The summation of a sequence. To understand the upcoming examples of instability, the way in which numbers are represented internally by microprocessors must be understood.

The following sections explain the cause of numerical stability and also illustrate the effect with real world computations. Whilst developing any numerically focused program, developers must bear in mind stability and the limitations of the floating point format. Whilst numerical stability may not be of great concern to programmers developing web tools or servers for text processing, any simulation work involving floating point representations used and reused in calculations must bear stability in mind. This is highlighted by a popular paper by Goldberg titled "What every computer scientist should know about floating-point arithmetic" [77].

Programs developed during the course of this thesis have been implemented with numerical stability in mind. Every care has been taken in ensuring that newly developed programs produce the same results on a range of platforms. To enable the in house virtual screening program LIDAEUS (Ligand Discovery at Edinburgh University) documented in chapter 4 to produce results consistent across a range of platforms, the numerically sensitive minimiser had to be examined. Its mode of operation was changed slightly so as to become less sensitive to numerical instability.

2.4.1 Numerical representation

In computing, it is important to represent a wide range of different number formats, each format built upon an underlying binary representation. Ignoring the more exotic imaginary number representations, there are two types of number of importance within computing. These are integers (whole numbers such as -2, -1, 0, 1, 2 and 4) and real numbers (such as 3.14159, 0.01 and 99998.001). The difficulty in expressing real numbers accurately lies not only in the range of the numbers that need to be stored, but also the level of precision that the number needs to retain. Standard number representations store a vast range of numbers in a fixed amount of storage (bits).

Providing a format which allows storing numbers such as 1.1 and 0.142857142857142857142857142857 (this is the irrational number 1/7) in a sensible number of bits proves difficult. A simple thought experiment can highlight

the difficulty present. Between the numbers 0.001 and 0.002 there lies an infinite amount of numbers. This infinite number space cannot therefore be represented in a fixed amount of storage. This leads to some numbers being impossible to represent (in a chosen fixed storage space). A fixed amount of storage leads to the quantisation of real number space.

There are three main approaches in computing to store real numbers.

2.4.1.1 Fixed point

Fixed point representation involves using a set number of bits to represent the integer part and the decimal part. Both parts being themselves fixed width integers. For example, storing 26.105 may take the form 0026 for the integer part and 1050 for the decimal. This representation has drawbacks and advantages. The range of numbers that can be represented relies entirely on how many bits (or the size of the integers used for each part) are used for the first and to a lesser extend the second part of the number (after the decimal point). A great advantage for this representation is that the accuracy with which the number is represented within the range for the format is exact down to the limit of digits in the second part. The example given above is a very high level implementation of fixed point representation.

2.4.1.2 Rationals

When using rationals, the real number is the ratio between two integers. For example, 26.105 may be stored in two parts, the first being 5221 and the second 200 (5221/200 = 26.105). This format offers great accuracy to certain irrational numbers, for example, one seventh cannot be expressed completely accurately without using an infinite number of digits if a fixed point representation is used. Using rationals, this number is easily and precisely expressed as 1 as the first number and 7 as the second, 1/7. Drawbacks present in the arithmetic of two numbers represented by rationals are obvious; the second part (denominators) must be the same for many numerical operations, introducing the requirement of further calculations.

2.4.1.3 Floating point

Floating point is the *de facto* standard of real number representation on microprocessors. As the name implies, the decimal point is floating within the representation and a part of the format specifies its position. Scientific number notation plays a big role in this representation. Scientific notation takes the form of a significand and an exponent. 26.105 in normalised scientific notation is represented as 2.6105×10^{1}. Here, 2.6105 is the significand and 1 the exponent showing that the decimal point should be moved one place to the right (2.6105×10^{1} can be treated as a simple expression, solving gives 26.105). The great advantage of scientific notation which is carried through to floating point is the range of numbers which can be represented. 0.000000000000001 and 1,000,000,000,000,000 can be represented as

$1x10^{-15}$ and $1x10^{15}$ respectively. When a floating point number is expressed to the user of a computer or other microprocessor device such as a calculator, the base of the exponent is omitted and replaced by ‘e’ or ‘E’. The exponent base is base 10 unless otherwise stated. The previous large and small number examples would therefore be expressed as 1e-15 and 1e15. Floating point representations typically come in two levels of precision, defined by the number of bits used to represent the number. The most common representation within microprocessors, the float data type, consists of 4 bytes which equates to 32 bits (there are 8 bits in a byte). The double data type represents real numbers using 8 bytes or 64 bits.

As a limited amount of precision is present, operations on the numbers often produce results which cannot be stored in the original representation. For example, 0.1 divided by 3.0 produces 0.03333, with 3 recurring infinitely. Further difficulty in floating point arithmetic is introduced by the need to define rounding. The rules for rounding and the entire floating point format are defined in the IEEE 754 standard from 1985 [78] and more recently a revision in 2008 [79] which amongst other things adds a new rounding mode.

Four rounding modes exist for floating point arithmetic, the default behaviour being “round to nearest’. In this case, an operation producing a potentially infinitely precise result is rounded to the nearest representable number. “Round up” rounds the result upwards towards positive infinity; negative numbers therefore round towards

zero. "Round down" rounds the result downwards towards negative infinity; negative numbers rounding away from zero, positive numbers towards zero. The final rounding mode is "round towards zero" where positive numbers are rounded down and negative numbers rounded up. As previously mentioned, the default behaviour is "round to nearest" although all IEEE 754 compliant hardware must support all rounding modes. An overview of floating point representation and arithmetic as described by the IEEE 754 standard is documented by Goldberg [77] in which the many problems present are explored.

Due to the nature of computation on microprocessors, the repetitive use of arithmetical operations compounds errors present. When considered on its own, the error from one calculation may not be significant; however, if this number is then operated upon many more times, each arithmetic step has the potential to add to the error present, amplifying it.

Even when this shortfall is accepted and assumed to be present in all computing calculations, researchers will often discover that running programs that operate on floating point numbers across different platforms and microprocessors produce different results. At first, this appears nonsensical as hardware adhering to the IEEE 754 standards should produce standard results. Whilst almost every area is covered in the standard, the internal action of microprocessors in performing arithmetic operations is not. The major reason for inconsistent results is a consequence of the

amount of guard digits used. A microprocessor reading in a single precision floating point numbers to be raised to the power two will read from RAM 4 bytes or 32 bits. The calculation on the microprocessor will be carried out in a greater precision than the standard representation of the number. It is common for microprocessors to make use of two guard digits. This equates to the 4 byte float being stored temporarily and being operated on in higher precision than its initial representation. The higher precision result must then be converted back to the size of the original representation and taken as the result. Microprocessors may use a different number of guard digits, depending on their hardware capabilities. Differences in the precision within which arithmetic is carried out between chips may cause rounding to be inconsistent. A greatly simplified example in base 10 would be if one microprocessor calculated the result to be 1.11 and the second 1.115. Rounding to two decimal places results in 1.11 and 1.12; the IEEE 754 may have been adhered to but results may differ. Inconsistent results from computation on different hardware are a major problem, one that has been addressed by the Java programming language. A seldom used option in Java can be enabled: ‘strictfp’ enforces strict floating point rules and forces computation to be carried out with no guard digits. Floating point operations are carried out in their default level of precision even if the microprocessor is capable of more precise results. This ensures consistent results across all platforms.

2.4.2 Herron's formula. Improving numerical stability with reordering of computation

The standard approach to calculating the area of a triangle is to use Herron's formula as follows:

$A = \sqrt{s(s-a)(s-b)(s-c)}$ where $s = \frac{a+b+c}{2}$ and *a*, *b* and *c* are the length of the sides of the triangle. Implementing this equation in the form above proves numerically unstable when the triangle is very slender. The equation can be rearranged into a numerically stable version of the form:

$$A = \frac{\sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}}{4} \text{ where } a \le b \le c .$$

A program has been written which uses both versions of Herron's formula (the unstable and stable versions) to illustrate the inaccuracies present in floating point arithmetic. The program uses 4 byte floats for calculation.

With *a*=12345679.0 and *b*=12345678.0 and changing values of *c* for the stable and unstable versions, the areas shown in Table 2.2 are calculated.

C	Area (unstable)	Area (stable)
400000.0	2.46881e+12	2.46881e+12
40000.0	2.46907e+11	2.46913e+11
4000.0	2.46852e+10	2.46914e+10
400.0	2.46296e+09	2.46913e+09
40.0	2.40662e+08	2.46836e+08
4.0	1.74594e+07	2.39073e+07
3.5	1.74594e+07	2.07043e+07
2	0	1.06917e+07
1.1	0	2.82875e+06
1.01	0	875151

Table 2.2 – Triangle areas obtained using numerically unstable and stable algorithms

From the results above, it is evident that the rearrangement of Herron's formula provides greater numerical stability, and results for a triangle of increasingly slender

proportions does not converge to zero as quickly as the unstable version. The stable version performs adequately with values of *c* greater than one.

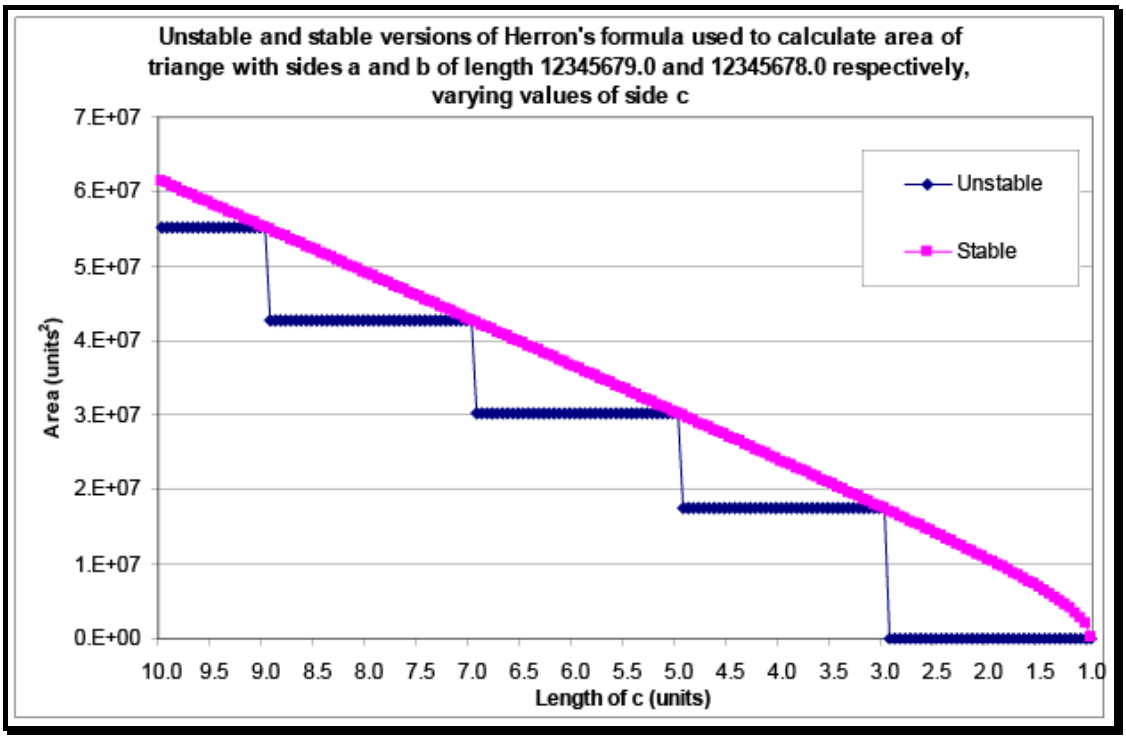


Figure 2.13 - Unstable and Stable versions of Herron's formula

Figure 2.13 shows the values produced by the unstable and stable versions of Herron's formula, with sides *a* and *b* equal to 12345679.0 and 12345678.0 respectively and *c* ranging from 10 to 1 units. Within this range, the unstable version has produced a stepping effect, indicating that results obtained from it are heavily influenced by an artefact of floating point representation. The results are quantised to certain values and results do not change linearly with the size of *c*; as appears to be occurring in the stable version. This effect is brought about by the quantisation present in IEEE 754 floating point representation; the quantised nature of a binary representation attempting to store a continuous dimension.

2.4.3 The summation of a sequence

A simple exercise to show the inaccuracy of floating point arithmetic on microprocessors is to calculate the summation of a sequence of numbers using different number representations.

The summation of the sequence of numbers 1 to 10,000 can be solved with a simple direct method using the following formula where n is the largest number in the range; in this case 10,000.

$$sum = \frac{n(n+1)}{2}$$

From this, the summation of 1 to 10,000 is 50,005,000

With the result known, a program was written in C++ to perform the summation using three different data types, integer, float and double. The program starts with the first iteration being 1 added to the running total, next iteration two adds 2 to the total and so on. Table 2.3 shows output at different iterations for each data type.

Iteration	Int	Float	Double
100	5050	5050	5050
1000	500500	500500	500500
2000	2001000	2.001e+06	2.001e+06
6000	18003000	1.80029e+07	1.8003e+07
7000	24503500	2.45029e+07	2.45035e+07
8000	32004000	3.20029e+07	3.2004e+07
9000	40504500	4.05029e+07	4.05045e+07
10000	50005000	5.00029e+07	5.0005e+07

Table 2.3 - The summation of all whole numbers 1 to 10,000 using different number formats

An interesting result also occurs when the float data type is used in the same way as above, but the calculation carried out in reverse, summing the numbers down from 10,000 to 1. In this case, the result is 5.00091e+07. This alone proves that with floating point arithmetic, $(a+b)+c$ is not equal to $(c+b)+a$. The integer data type is obviously the correct choice for the goal of this program, providing the correct answer, as does the double data type. Float however becomes more and more inaccurate as larger numbers are added; less priority is given to the bits representing

small parts of the number. As the summation continues, the small error increases. The same effect is not seen using the double data type which uses twice the number of bits to represent a number than the float data type. However, as the encoding of floats has been proved not to be entirely accurate, one can see how doubling the number of bits used in the representation of a number ultimately increases accuracy, but does not make the representation one hundred percent accurate. With enough iterations the error would grow noticeably large.

2.5 Algorithms

Computer code written during the course of this research and tool development makes use of the algorithms described in this section. An overview of algorithms deemed important and key to virtual screening techniques are described.

2.5.1 Root Mean Squared Deviation

Root Mean Squared Deviation (RMSD) can be used to express the differences present in a set of points arranged in two different ways. In biochemistry, it is typically used when measuring the differences found in two representations of the same protein. A conformational change in protein structure can be quantified using RMSD. A ligand binding to a protein may cause a change in the conformation of the protein which can then be quantified. Figure 2.14 shows two molecules overlayed so as to minimise the RMSD between them.

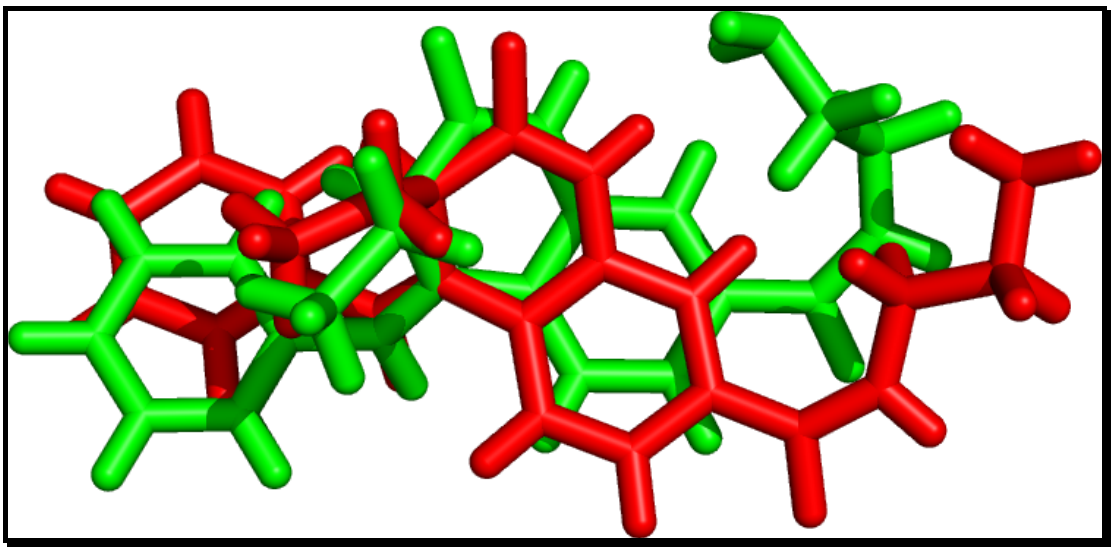


Figure 2.14 - A molecule in two different conformations, RMSD between the two conformations is 1.74Å. All atoms except hydrogens were taken into consideration for the RMSD calculation.

RMSD is given by:

$$RMSD = \sqrt{\frac{\sum_{i=1}^{N_{atoms}} d_i^2}{N_{atoms}}}$$

80

where N_{atoms} is the number of atoms/points being matched and d is the distance between corresponding points in each set.

Calculating d^2 is achieved, getting the squared distance between two points in the following way:

$$d_i^{\,2} = (s1.i.x - s2.i.x)^2 + (s1.i.y - s2.i.y)^2 + (s1.i.z - s2.i.z)^2$$

Where s1 is set 1, s2 set 2, i is the point within the sets being compared, and x, y and z refer to the x, y and z coordinates of the points.

An example is shown below using two simple sets of three points in 3D space.

The two sets contain three points each, each point occupying a known position in 3D orthogonal space as given by its x, y and z coordinate. Point 1 in set 1 matches to point 1 in set 2 and similarly for the remaining points in each set.

	Set 1					Set 2		
	p1	p2	p3			p1	p2	p3
x	1.1	1.3	1.8		x	1.2	1.5	2.1
y	-3.9	1.9	2.1		y	-3.0	2.1	2.3
z	4.2	1.8	2.9		z	3.8	1.4	3.3

Table 2.4 – 3D coordinates of 2 sets of points

To calculate the RMSD, the distance between the points p1 in set 1 and 2 (as given in Table 2.4)

is calculated,

$$D = \sqrt{\begin{array}{l}(Set1Point1x - Set2Point1x)^2 + \\ (Set1Point1y - Set2Point1y)^2 + \\ (Set1Point1z - Set2Point1z)^2\end{array}}$$

This is then carried out for points 2 and 3 from each set. Using the above data,

$d_1^2 = 0.14 \text{ units}^2$

$d_2^2 = 0.89 \text{ units}^2$

$d_3^2 = 0.48 \text{ units}^2$

Therefore, the RMSD between these two sets of points is

$$RMSD = \sqrt{\frac{0.14 + 0.89 + 0.48}{3}} = 0.709 \text{ units}$$

RMSD is not limited to small molecules. Fitting representations of proteins together can also be achieved. Using the PyMol [43] molecular graphics package, two structures can be superimposed by way of direction from the minimisation of RMSD between a set of chosen points. The protein FKBP12 (See Chapter 1) is represented in two PDB entries with ID numbers 1FKJ and 2DG3. Visualising these structures in PyMol shows that they are not in the same frame of reference as shown in Figure 2.15. The two structures are shown aligned in Figure 2.16 with a calculated RMSD of 0.5Å.

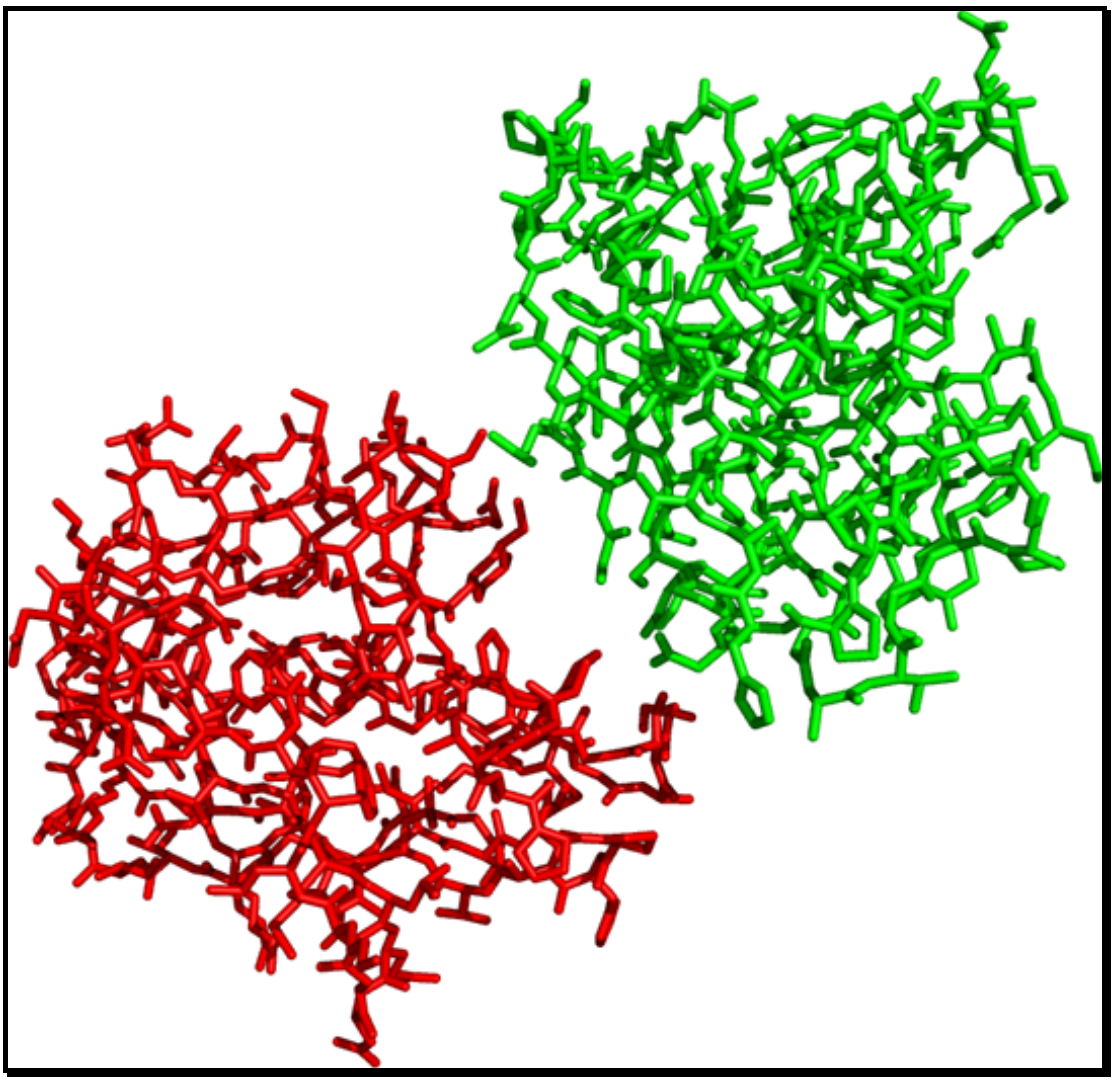


Figure 2.15 – Non aligned 1FKJ (shown in green) and 2DG3 (shown in red) visualised together in space

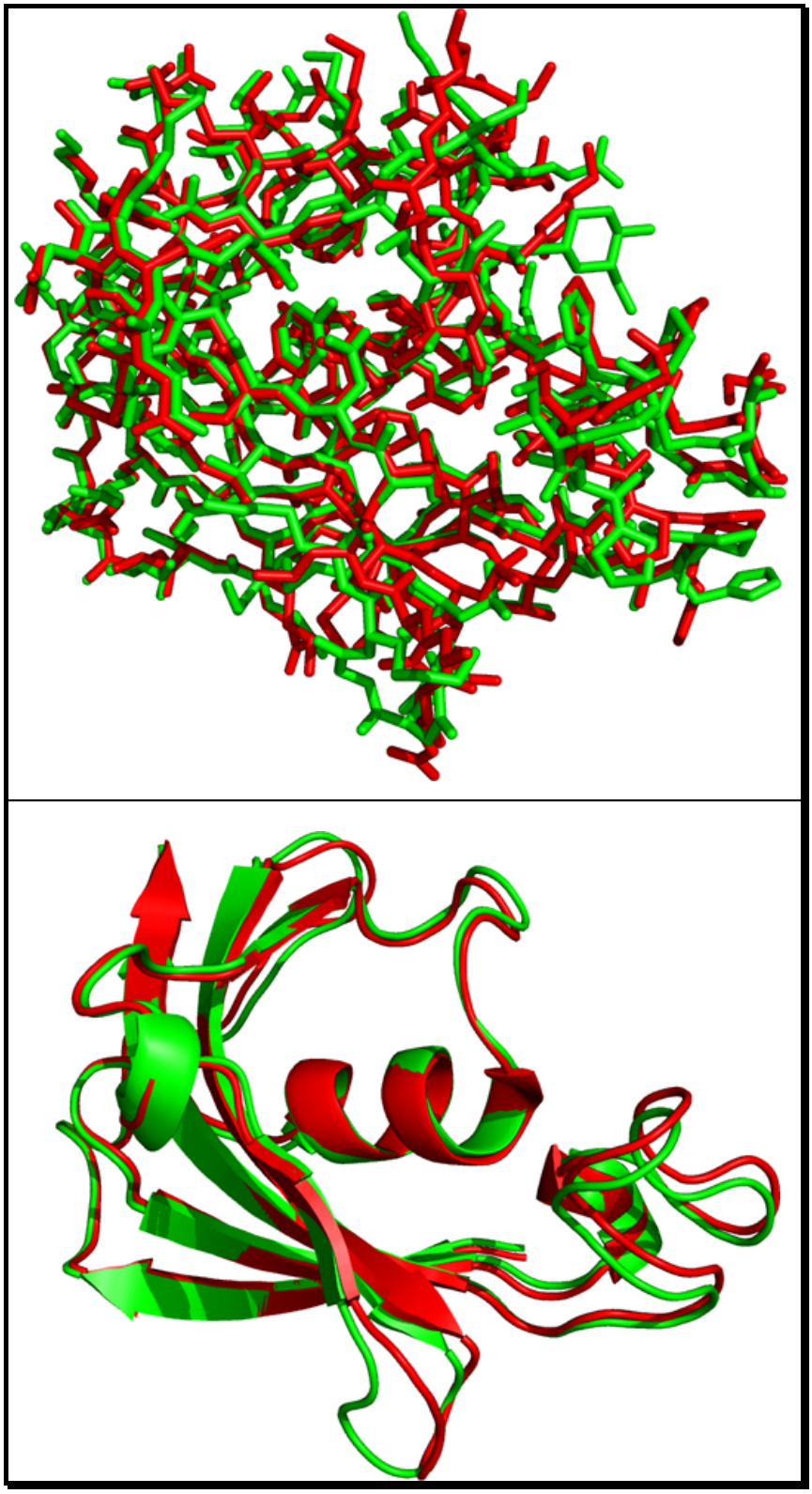


Figure 2.16 – FKBP12 aligned from 1FKJ (shown in green) and 2DG3 (shown in red). RMSD between the two structures (calculated from position of carbon alphas) is now 0.5Å. Stick representation above, cartoon below

2.5.2 Generating rotation matrices

Overlaying an ordered set of points on to another requires the generation of a rotation matrix. If points do not overlay exactly then there is a residual. The optimum matrix will overlay the set of points in such a way as to have the minimum possible residual. The technique outlined by Kearsley [81] is used for this purpose.

2.5.3 Point matching

Point matching is an important concept in the placement of ligands into complex with proteins. Many virtual screening codes start their positioning with a rough placement of the candidate ligand into the binding site of interest. In the case of the virtual screening code LIDAEUS, described in Chapter 4, point matching is carried out with the goal of overlaying ligand atoms onto predetermined energetically favourable points called site points. To achieve this, the positions of atoms within the candidate ligand are examined. A set of distance of all atoms to all other atoms is then generated. With this set, the spatial distribution of ligand atoms is described. The same is done for the list of site points which are treated in exactly the same way as the candidate ligand atoms. With these two sets generated, the matching procedure can begin. A standard LIDAEUS run aims to overlay four ligand atoms onto four site points. This is then considered a pose. This overlay can only be achieved through translation and rotation of the ligand represented as a rigid body. To test if an overlay such as this is possible (within tolerances), a recursive depth first search of all possibilities is carried out. In this example, we will ignore the fact that only certain atoms can overlay their complimentary site point counterparts and focus on just matching points in space. First, a ligand atom is chosen and the list of

distances to all other ligand atoms is examined. If any of the distances are within the predefined tolerance for point matching, then this set of two atoms can be kept track of. The process continues; the algorithm looking for a common distance of the point in each set. If found, the process continues to a fourth atom. At each stage, when a match is found, other matches may also exist; these are also taken into account. If a match on the path one position beyond the position taken note of is a dead end, the algorithm returns to a level at which there were other possibilities. With many possible overlays of four atoms generated, the next stage is to determine a rotation matrix which, when applied to the candidate ligand, will overlay its chosen atoms with correspondingly chosen site points. This is a trivial task which uses the mathematical approach outlined by Kearsley and detailed in section 2.5.2 Generating rotation matrices. With at least three points in 3D space that are not arranged in a problematic way, such as a straight line or all at the same place in space, it is possible to calculate a unique rotation which, when applied, will overlay the required points. The usual operation of LIDAEUS is to overlay four points.

2.5.4 Force field energy calculation

In biochemistry, force fields are used to give a measure of the energy in a system. A single energy value for a system is not useful in itself, but when compared against the energy of the system in another state, the difference between the two values can be used to determine the most energetically favourable state. Chemical force fields are defined in terms of parameters for objects of a certain type and in a certain state. An equation takes into account these values and other properties of the system, resulting in a predicted energy. Comparing two energy states therefore gives the energy difference. Exploring different states offers the ability to search for an energetically favourable arrangement.

A simple extract of the Tripos [82] force field along with a small simplified example using two force field terms is given below.

Suppose we have a water molecule, H_2O. Looking up values from the Tripos forcefield shows us that the ideal single bond length between an O.3 and an H atom is 0.95 Ångstroms, and also has another constant associated with it. With these figures and knowing the conformation and atom positions of the water molecule in 3D space, the bond lengths can easily be calculated and values used in the following equation to obtain a measure of the energy contribution made by the bond lengths to the internal energy of the molecule.

$$bondE = \sum_{i=1}^{n} \frac{1}{2} k_b (l_i - lo_i)^2$$

Where *bondE* is the energy contribution from bond length deviations from their optimum lengths to the system and n is the number of bonds in the system.

Further calculations can then be carried out, such as the one below which puts an energy on the angle formed by the three atoms in the water molecule.

$$angleE = \sum_{i=1}^{n} \frac{1}{2} k_a (\theta - \theta_0)^2$$

Where *angleE* is the enegy contribution from bond angles, summing over the set of bond angles present in the molecule with θ denoting the observed bond angle, θ_0 the ideal bond angle and k_a a constant associated with a bond angle between 3 atoms of the appropriate type being measured.

A simple force field has been implemented in an attempt to incorporate flexible ligand docking into the in-house structure based virtual screening code LIDAEUS. This is detailed in Section 4.5 Simple flexible docking in LIDAEUS.

3 Manipulating molecular information

The representation of molecular information can take many formats. This chapter will go through a number of well known and relevant storage and representation methods. Molecular data usually encodes atom positions and bond information. Details on conformation generation and filtering are also given.

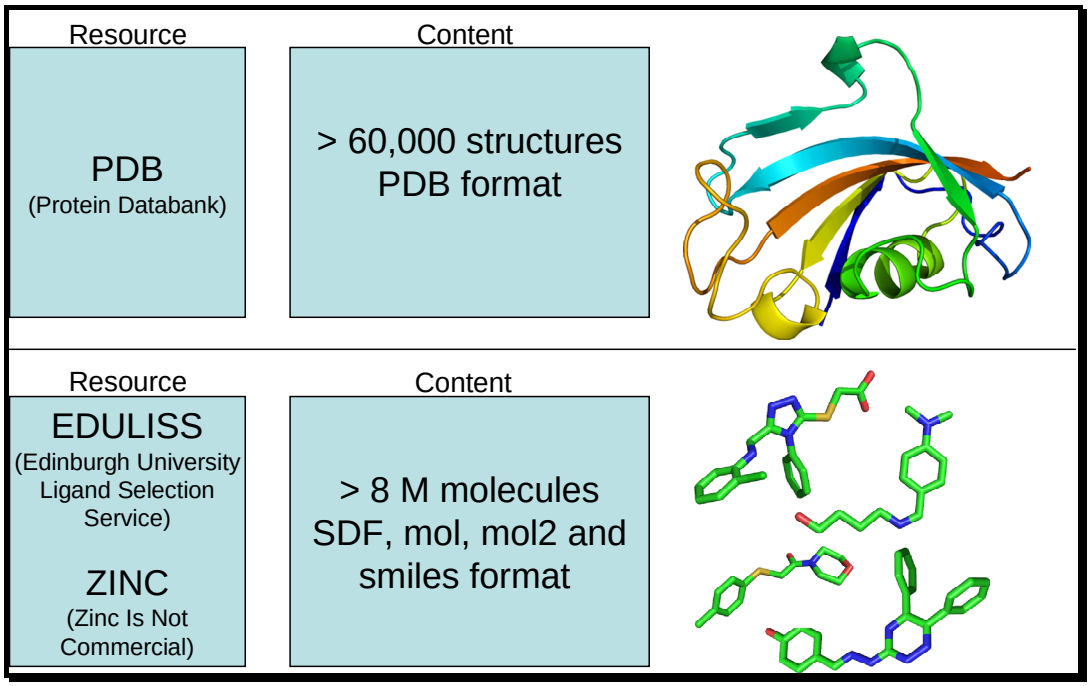


Figure 3.1 - The data sources and scale of molecules represented by different molecular information formats

3.1 Molecular File Formats

With literally hundreds of file formats existing for the representation of molecular data, it can be difficult for researchers to ensure their data is usable across a range of programs and platforms. Often, conversion of formats is required. Whilst many molecular modelling, creation and visualisation packages such as ChemSketch, ChemDraw and PyMol [43; 83; 84] allow the saving of data in many representations, it is often advantageous to use a standalone program to achieve conversion between types. The most common and well known molecular file format converter is OpenBabel [85; 86]. It is open source software and available on many platforms. The most common way it is used is through the command line, although versions with a graphical user interface are available. As of version 2.2.0, OpenBabel allows the conversion to and from 97 different molecular representations.

Throughout this report, two main types of file format are used to represent molecules. Structure-data file (SDF) and Protein Data Bank (PDB). SDF files typically represent small molecules whilst PDB is concerned with large macromolecules such as proteins. One reason for this is the restrictive fixed formatting of the SDF format, allowing the maximum number of atoms contained within a molecule to be 256. An important step in using the LIDAEUS virtual screening program discussed later in Chapter 4 requires the use of a mol2 file. A description of the format of mol2 files is available in section 3.1.5 The mol2 file format. A discussion of the SMILES format is also given.

3.1.1 The SDF file format

In describing the Structure-Data File (SDF) format, its lineage must be considered. Developed by Molecular Design Limited, it is part of the Chemical Table (CT) file family. From the original concept of a connection table (Ctab), other formats have been created. The Ctab is a table documenting atoms and the bonds between atoms.

Figure 3.2 shows two members of the CT file format, MOL and SDF.

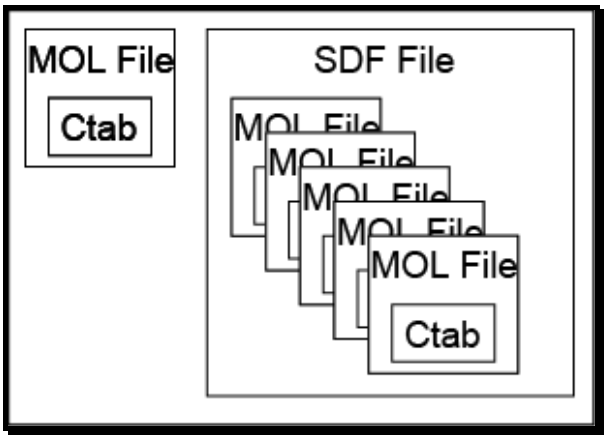


Figure 3.2 – MOL and SDF files, members of the CT file family

Adapted from [87]

A full description of the CT-based file formats is presented in various papers [87; 88] and will not be given here.

Figure 3.3 shows a small molecule represented in the SDF file format (line numbers given for explanation clarity). Figure 3.4 shows a 3D representation of this molecule.

```
 1 875760
 2   DSViewer           3D                              0
 3
 4  17 18  0  0  0  0  0  0  0  0999 V2000
 5    -1.3389    0.0001   -0.7000 C   0  0  0  0  0  0  0  0  0  1
 6    -0.0235    0.0001    0.0413 C   0  0  0  0  0  0  0  0  0  2
 7    -0.0086    0.0460    1.4403 C   0  0  0  0  0  0  0  0  0  3
 8     1.2111    0.0460    2.1277 C   0  0  0  0  0  0  0  0  0  4
 9     2.4157    0.0000    1.4160 C   0  0  0  0  0  0  0  0  0  5
10     2.4007   -0.0459    0.0168 C   0  0  0  0  0  0  0  0  0  6
11     1.1811   -0.0458   -0.6704 C   0  0  0  0  0  0  0  0  0  7
12     3.9577    0.0000    2.2850 S   0  0  0  0  0  0  0  0  0  8
13     4.4831   -1.7159    2.5883 C   0  0  0  0  0  0  0  0  0  9
14     5.7908   -1.7159    3.3433 C   0  0  0  0  0  0  0  0  0 10
15     6.4367   -0.6751    3.4546 O   0  0  0  0  0  0  0  0  0 11
16     6.2442   -2.8413    3.8880 N   0  0  0  0  0  0  0  0  0 12
17     7.3927   -2.8713    4.8049 C   0  0  0  0  0  0  0  0  0 13
18     8.3589   -4.0270    4.4845 C   0  0  0  0  0  0  0  0  0 14
19     7.6652   -5.2737    4.3871 O   0  0  0  0  0  0  0  0  0 15
20     6.6795   -5.2526    3.3512 C   0  0  0  0  0  0  0  0  0 16
21     5.6359   -4.1525    3.6201 C   0  0  0  0  0  0  0  0  0 17
22   1  2  1  0  0  0
23   2  3  2  0  0  0
24   3  4  1  0  0  0
25   4  5  2  0  0  0
26   5  6  1  0  0  0
27   6  7  2  0  0  0
28   7  2  1  0  0  0
29   5  8  1  0  0  0
30   8  9  1  0  0  0
31   9 10  1  0  0  0
32  10 11  2  0  0  0
33  10 12  1  0  0  0
34  12 13  1  0  0  0
35  13 14  1  0  0  0
36  14 15  1  0  0  0
37  15 16  1  0  0  0
38  16 17  1  0  0  0
39  17 12  1  0  0  0
40 M  END
41 > <Outcome>
42 Active
43
44 > <PUBCHEM_IUPAC_NAME>
45 2-(4-methylphenyl)sulfanyl-1-morpholin-4-yl-ethanone
46
47 $$$$
```

Figure 3.3 - A small molecule represented in the SDF format

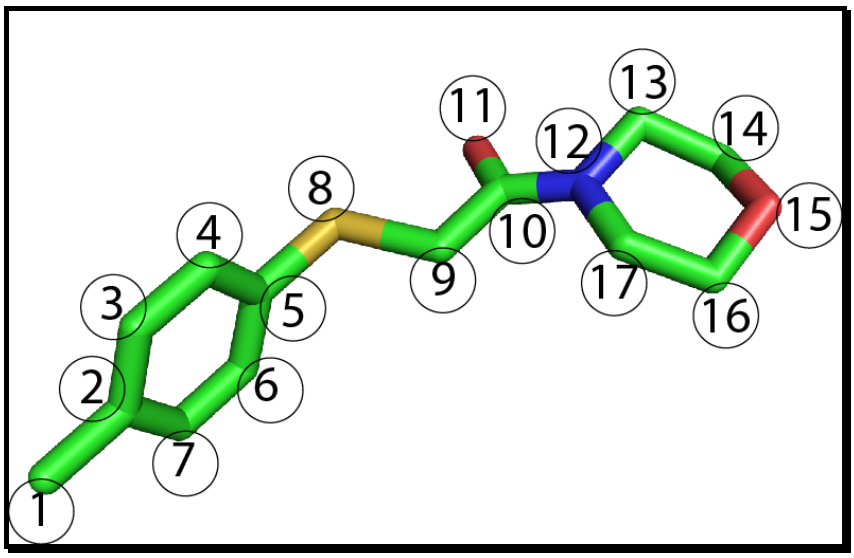


Figure 3.4 - Small molecule represented in the SDF format (given in Figure 3.3) rendered as sticks with atom numbers attached

The SDF file format is essentially a way to represent multiple molecules encoded into the MOL format in a single text file. Molecules in the SDF file format are essentially molecules expressed in the MOL format with the addition of a separator between molecules. Line 1 is the title line. Here, the title contains an identification number (875760); lines 2 and 3 are comment lines. Line 4 contains first the number of atoms in the molecule and the number of bonds present between these atoms. Lines 5 through to 21 contain atom information giving 3D x, y and z coordinates for the position of the centre of the atoms. A letter then denotes the type of atom at this point. Line 22 to 39 represents the connection table. A line in the connection table first identifies two atoms (atom order is taken from the atom information above, the first atom will have the ID 1, the second 2 and so on). After identifying two atoms bound together, a bond order is given, i.e. single bond or double bond (1 or 2). The end of the connection table which ran from line one is denoted by line 40 which must

read: “M END”. Lines in the representation after this point can be considered supplementary information on the molecule. This information takes the form of ‘keys’ which are of the form:

```
>  <TITLE>

Information line 1

Information line 2
```

Multiple information lines may be present in a key and the end of a key represented by an empty line. Lines 41 to 42 and 44 to 45 represent two keys in figure 2. Line 47 represents the end of the representation of a molecule. After this marker, another molecule may follow.

3.1.2 The SDF Toolkit

Throughout the period of research, a number of tools have been written to make dealing with and manipulating SDF files easier, all designed to be run from the LINUX command line. All of these small programs have formed a simple toolkit essential for dealing with SDF files during the course of research.

The toolkit allows the following actions to be carried out on SDF files

- SDF entry extraction by number, range or identifier
- Random extraction of molecules for creation of test/benchmarking datasets
- Shuffling the order in which the molecules appear in an SDF file
- Counting the number of occurrences of a molecule in an SDF file
- Insertion of identity matrix keys

The framework makes extensive use of the code used by the virtual screening program LIDAEUS (described in Chapter 4)

3.1.2.1 ChopSDF

A tool to extract specific sections of an SDF file.

Example usage:

```
chopsdf input.sdf 100 200 > output.sdf
```

Explanation: Running the above command would mean that the SDF file 'input.sdf' would be read in, and the 100th to 200th molecules appearing in the file written to the newly created file 'output.sdf'. Behaviour of the program in unforeseen circumstances is reasonable with sensible bounds checking. Asking for the minus 100th to positive billion molecules in a file when there only exists two molecules in the input SDF will supply the molecules existing within this range (molecules one and two).

3.1.2.2 ExtractMols

Often a set of molecules with known titles need to be extracted from a larger SDF.

Example usage :

```
extractmols molstokeep.txt < input.sdf >extractedmols.sdf
```

The above command invokes the extractmols program to read in a text file ('molstokeep.txt') containing the title fields of the molecules to extract, each separated onto a new line. If the title of any requested molecule is found in the file 'input.sdf' then it is written out to 'extractedmols.sdf'

3.1.2.3 GrabRandom

This program extracts a number of randomly selected molecules from an SDF file. Example usage:

```
grabrandom 100 < input.sdf > random100.sdf
```

The above command would extract 100 molecules at random from the SDF file 'input.sdf' and store the results in 'random100.sdf'. This tool can be useful in creating test datasets where a random selection of molecules is required for testing purposes. The random number generator used is the pseudo random number generator included with the GNU compiler suite, making use of a call to 'rand', the generator seeded with the number of milliseconds elapsed since midnight on January 1st 1970.

3.1.2.4 ShuffleSDF

This program is an extension of the grabrandom program which shuffles the position of molecules within an SDF file.

Example usage:

```
shufflesdf < input.sdf > shuffled.sdf
```

The above command invokes the shufflesdf program to read in 'input.sdf' and write a version where the positions of molecules within the file have been shuffled. Similar to the grabrandom program, shufflesdf makes use of calls to 'rand' for random number generation.

3.1.2.5 CountOccurrences

Similar in operation to the above extractmols program, and following almost the same syntax.

Example usage:

```
countoccurrences querytitles.txt < input.sdf
```

The above command invokes countoccurances to read in ‘querytitles.txt’ a text file containing molecule titles separated by new lines. The program will read in ‘input.sdf’ and keeps track of the number of molecules with titles matching entries in ‘querytitles.txt’. After completing a search for matching molecules the total count is output.

3.1.2.6 IdentInjector

The operation of the virtual screening program LIDAEUS (detailed in Chapter 4) upon molecules operates upon keys added to the end of the molecule’s representation within SDF files after the CTab. In some situations, especially during testing, the need to insert a four by four identity matrix as a key is often encountered. This program fulfils this purpose. Usage:

```
identinjector < input.sdf > output.sdf
```

The above command will invoke identinjector to read in ‘input.sdf’ and append a key containing an identity matrix of the form:

```
>  <MATRIX_INFO>

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1 0
```

The above is an identity matrix with the addition of one extra value in the final line. This number is required by LIDAEUS, and in normal function, denotes the root mean squared deviation produced by applying this combined rotation and translation matrix to the molecule. It is a measure of fit to site points which the program aims to overlay the molecule onto.

3.1.3 The PDB file format

The Protein Data Bank (PDB) format is suited to the representation of large macromolecules primarily due to the hierarchical description employed. Usually representing proteins, PDB allows a massive amount of structural data to be represented conveniently. PDB files can be many megabytes in size and represent multiple proteins in complex with themselves, DNA, RNA, other biochemical molecules and also their interactions with solvent. The PDB file format is heavily associated with the Protein Data Bank [15; 16; 17] as described in the first chapter of this report. The submission of structural information by researchers into the Protein Data

Bank is through the use of PDB files. These files represent the structures solved using a variety of techniques, such as X-ray crystallography or nuclear magnetic resonance (NMR). The most up to date specification for the format is freely available [89] and can act as guide to aid understanding for users, and act as a guide to developers creating programs that use the format. The components of a typical PDB file are given below. Whilst the PDB format can contain a lot of information, often in the form of remarks, the Protein Data Bank has minimum requirements for acceptance into its database which are stricter than is necessary for a valid PDB file.

Most records in the PDB standard deal with fixed width data. Line one is represented in Figure 3.5 with column numbers to aid description of the fixed width format used by ATOM records.

Figure 3.5 - An ATOM record entry from Protein Data Bank entry with PDB ID 2DG3, addition of column numbering to demonstrate the fixed width formatting required.

Columns 1 to 6 are used to define the record type. Within columns 7 to 11 is an atom serial number; this uniquely defines each atom in the PDB file, and essentially is a running total of the number of atom entries within the file. Columns 13 to 16 give an atom name, in this case ‘N’. Columns 18 to 20 give the three letter code of the residue being represented, in this case, GLY which is the natural amino acid glycine. Column 22 gives a character representing the chain that the residue belongs

to. Chains in PDB files are described using letters, a dimer will have two chains; ‘Chain A’ and ‘Chain B’. Columns 23 to 26 give the sequence number of the residue which the atom belongs to. In the PDB file, each residue is made up of multiple atoms. The grouping of these atoms into residues can be achieved through matching of the residue sequence number. In the more extensive Figure 3.6, the residue number can be seen to be increasing as new residues are represented.

A minimal representation with enough information for a small structure to be represented is shown below. Three residues from the protein FKBP12 submitted to the Protein Data Bank with structure id 2DG3 are represented as follows:

```
1  ATOM      1  N   GLY A   1      15.916  -2.311   8.217  1.00 30.78           N
2  ATOM      2  CA  GLY A   1      14.902  -1.669   7.330  1.00 30.25           C
3  ATOM      3  C   GLY A   1      13.472  -1.945   7.754  1.00 30.23           C
4  ATOM      4  O   GLY A   1      13.120  -1.785   8.925  1.00 30.32           O
5  ATOM      5  N   VAL A   2      12.645  -2.342   6.791  1.00 29.83           N
6  ATOM      6  CA  VAL A   2      11.257  -2.719   7.059  1.00 29.64           C
7  ATOM      7  C   VAL A   2      10.926  -4.057   6.393  1.00 29.58           C
8  ATOM      8  O   VAL A   2      11.135  -4.234   5.189  1.00 29.54           O
9  ATOM      9  CB  VAL A   2      10.244  -1.598   6.636  1.00 29.65           C
10 ATOM     10  CG1 VAL A   2      10.404  -1.215   5.161  1.00 29.50           C
11 ATOM     11  CG2 VAL A   2       8.804  -2.010   6.943  1.00 29.31           C
12 ATOM     12  N   GLN A   3      10.443  -5.002   7.195  1.00 29.36           N
13 ATOM     13  CA  GLN A   3       9.942  -6.269   6.681  1.00 29.53           C
14 ATOM     14  C   GLN A   3       8.430  -6.182   6.577  1.00 29.04           C
15 ATOM     15  O   GLN A   3       7.764  -5.715   7.502  1.00 29.12           O
16 ATOM     16  CB  GLN A   3      10.342  -7.432   7.592  1.00 29.56           C
17 ATOM     17  CG  GLN A   3      10.109  -8.807   6.965  1.00 30.50           C
18 ATOM     18  CD  GLN A   3      10.213  -9.961   7.950  1.00 30.47           C
19 ATOM     19  NE2 GLN A   3      10.580  -9.663   9.192  1.00 31.78           N
20 ATOM     20  OE1 GLN A   3       9.966 -11.112   7.589  1.00 32.43           O
21 ATOM     21  N   VAL A   4       7.897  -6.633   5.446  1.00 28.65           N
22 ATOM     22  CA  VAL A   4       6.461  -6.599   5.195  1.00 28.28           C
23 ATOM     23  C   VAL A   4       5.939  -8.029   5.075  1.00 28.06           C
24 ATOM     24  O   VAL A   4       6.353  -8.779   4.189  1.00 28.00           O
25 ATOM     25  CB  VAL A   4       6.125  -5.782   3.919  1.00 28.36           C
26 ATOM     26  CG1 VAL A   4       4.627  -5.783   3.651  1.00 28.11           C
27 ATOM     27  CG2 VAL A   4       6.645  -4.346   4.043  1.00 28.34           C
28 END
```

Figure 3.6 - PDB file containing 4 residues from the protein FKBP12, taken from Protein Data Bank entry with PDB ID 2DG3.

The PDB is made up of lines, each starting with a ‘record name’. In Figure 3.6, two different record types are displayed, one for atom entries and one denoting the end of the PDB file. Atom entries as well as all other records follow strict rules laid out and available in the PDB file format specification [89]. Looking at Figure 3.6, which has line numbers added to the left of the contents for explanation clarity, line one represents an atom entry. Figure 3.7 shows the protein FKBP12 (entry 2DG3 in the protein Data Bank) from which the above example PDB file was generated. The representation is of the entire protein rendered as sticks, overlaid with a cartoon

representation and also showing the four residues from which the example above is taken in stick representation.

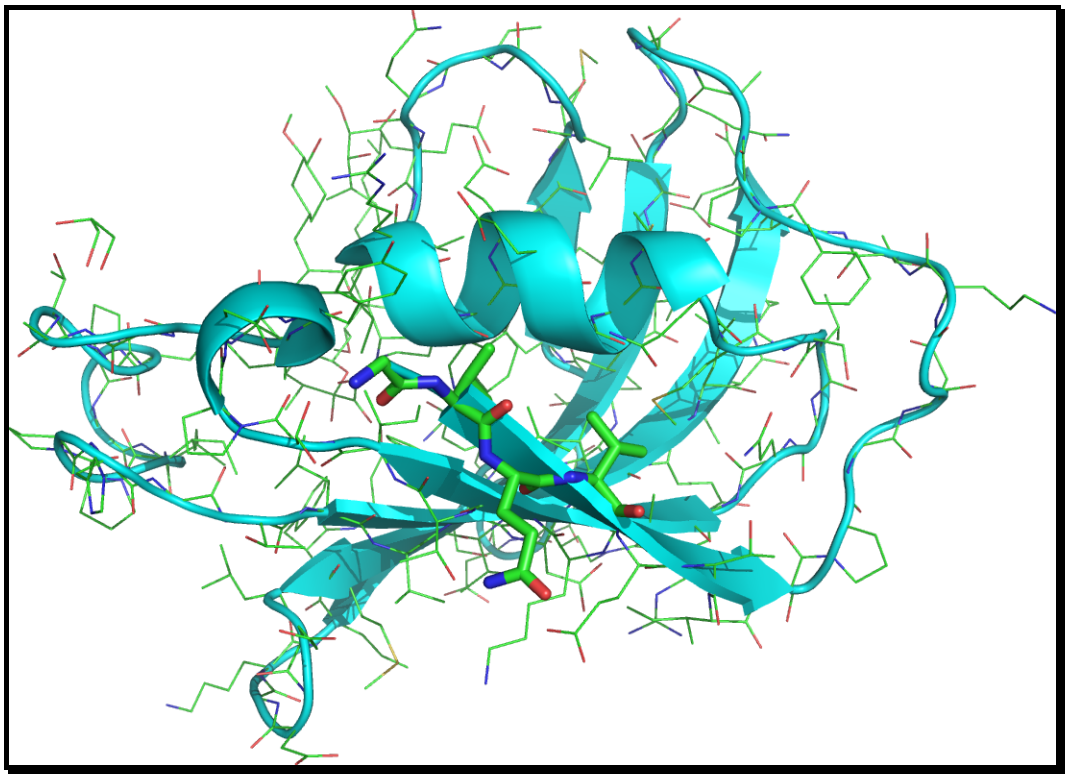


Figure 3.7 - FKBP12 (PDB ID 2DG3) represented as lines, cartoon shown and 4 residues from PDB example file shown with a stick representation.

3.1.4 The SMILES format

Simplified Molecular Input Line Entry System (SMILES) [90; 91] is a way of describing a molecule based purely on connectivity. A molecule described by smiles is simply a string of characters. This means that SMILES is not a fully fledged file format, a SMILES file being just a text file with ASCII strings representing molecules, one molecule per line. A simple smiles string representing carbon dioxide may be written as: OCO or equally, C(O)O. From this example, the structure of SMILES strings can be seen; atoms of organic nature are represented by their symbol. These organic atoms are boron, carbon, nitrogen, oxygen, phosphorus, sulphur, fluorine, chlorine, bromine, and iodine. All other atoms should be enclosed in square brackets; for example, '[Na]' for sodium. In the carbon dioxide example, the string is read from left to right, starting with an oxygen atom that is bound to a carbon, we can see that the carbon is then again, bound to an oxygen atom.

Rings can be expressed in this notation using numbers to show where a cut in the graph of the molecule has been made to reduce it to a linear notation as used in SMILES representations. Benzene is represented as follows: 'c1ccccc1'. Notice that lower case characters are being used for carbon. Atoms in lowercase denote aromaticity. 'c1ccccc1' states that the first carbon has been given an index of '1'. The sixth carbon is then shown to be bound to carbon one by inclusion of the index number.

Branching is achieved through the use of parentheses using standard brackets. In SMILES representations, unless otherwise specified, the order of bonds between atoms is assumed to be single. Double bonds can be specified with the addition of the '=' sign between bound atoms. Figure 3.8 shows a Vitamin C, or L-ascorbate molecule, where atom symbols are given with atom numbers (excluding carbon, in this case, only the atom number is given), and counted as they appear in the accompanying SMILES representation. The example shows how branching is achieved through the use of parentheses. Colour coding of two nested regions within the SMILES string is shown for easier association.

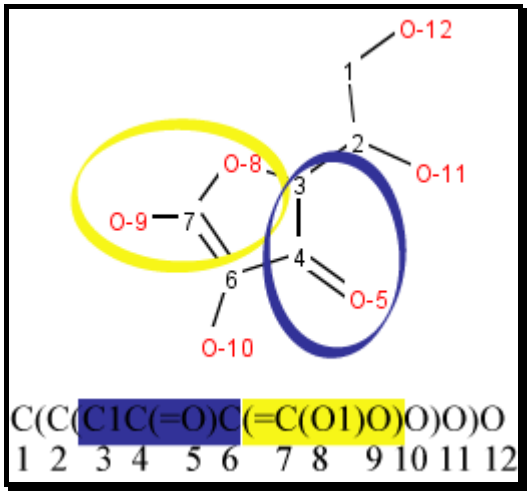


Figure 3.8 - Vitamin C with SMILES representation colour coded to aid understanding of SMILES representation. Atoms represented with one letter element code and atom number as appears in SMILES representation.

Support for chirality and stereochemistry is present, the characters '/' and '/' surrounding atoms bonded with a double bond implies trans-isomerism. Similarly, '/' and '\' implies cis-isomerism. Tetrahedral carbon configurations can also be specified.

Isotopes of atoms can also be defined by prefixing the atom symbol with a number denoting the mass of the isotope. For example, 'heavy water' can be defined as follows:

'[2H]O[2H]'.

As with the example on carbon dioxide, it is often the case that molecules can be expressed in SMILES notation in a variety of ways. A newer and more up to date form of the SMILES representation is canonical SMILES. Many algorithms exist to create canonical SMILES from many other formats (such as SDF or mol2) and this representation ensures that a molecule can only be represented in one way. This is of great use in many situations, such as searching a database or other repository to see if it contains a target compound; with non canonical SMILES, every possible permutation would have to be searched.

Figure 3.9 shows some common compounds represented as SMILES strings and graphical representations of their 2D structures.

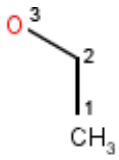
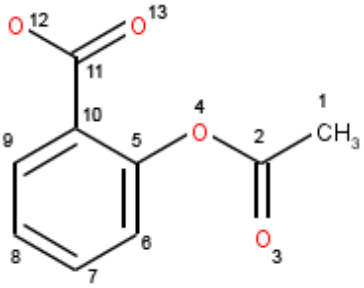
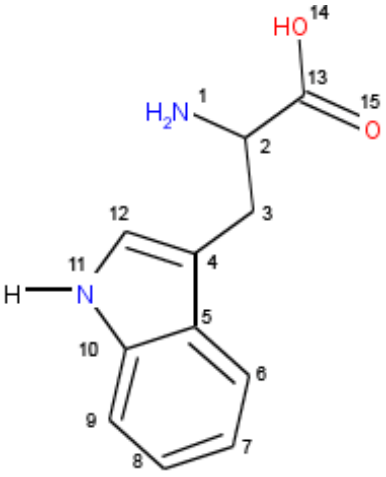
Common name & SMILES	2D representation
Ethanol CCO 1. 2 .3	
Aspirin CC(=O)OC1=CC=CC=C1C(O)=O 1. 2. 3. 4. 5. 6.7. 8. 9. 10. 11. 12. 13.	
Tryptophan N[C@@H](Cc1c2ccccc2n([H])c1)C(O)=O 1. 2. 3.4. 5. 6-- 10. 11. 12. 13.14. 15.	

Figure 3.9 - Ethanol, aspirin and tryptophan represented as SMILES strings along with their 2D molecular depictions

3.1.5 The mol2 file format

Although the name mol2 implies an extension of the MDL mol format, the Tripos mol2 format is syntactically unrelated. Figure 3.10 shows a representation of benzene, taken from the Tripos mol2 format definition [92].

```
#       Name: benzene
#       Creating user name: tom
#       Creation time: Wed Dec 28 00:18:30 1988

#       Modifying user name: tom
#       Modification time: Wed Dec 28 00:18:30 1988

@<TRIPOS>MOLECULE
benzene
12 12 1 0      0
SMALL
NO_CHARGES


@<TRIPOS>ATOM
1      C1     1.207  2.091  0.000  C.ar   1      BENZENE0.000
2      C2     2.414  1.394  0.000  C.ar   1      BENZENE0.000
3      C3     2.414  0.000  0.000  C.ar   1      BENZENE0.000
4      C4     1.207  -0.697 0.000  C.ar   1      BENZENE0.000
5      C5     0.000  0.000  0.000  C.ar   1      BENZENE0.000
6      C6     0.000  1.394  0.000  C.ar   1      BENZENE0.000
7      H1     1.207  3.175  0.000  H      1      BENZENE0.000
8      H2     3.353  1.936  0.000  H      1      BENZENE0.000
9      H3     3.353  -0.542 0.000  H      1      BENZENE0.000
10     H4     1.207  -1.781 0.000  H      1      BENZENE0.000
11     H5     -0.939 -0.542 0.000  H      1      BENZENE0.000
12     H6     -0.939 1.936  0.000  H      1      BENZENE0.000
@<TRIPOS>BOND
1      1      2      ar
2      1      6      ar
3      2      3      ar
4      3      4      ar
5      4      5      ar
6      5      6      ar
7      1      7      1
8      2      8      1
9      3      9      1
10     4      10     1
11     5      11     1
12     6      12     1
@<TRIPOS>SUBSTRUCTURE
1      BENZENE1      PERM   0      ****   ****   0      ROOT
```

Figure 3.10 – A representation of benzene in the mol2 file format, reproduced from the Tripos mol2 format definition [92].

The representation makes use of records as detailed below. Many records exist which are not documented here. For a full definition, the format guide is available [92].

Lines 1 to 7 are essentially comment lines, with any text present prefixed with the hash character, denoting a comment to be ignored.

The first record to appear is '@<TRIPOS>MOLECULE', starting at line 8. This record contains information that relates to the whole molecule. Data belonging to this 'MOLECULE' record is derived as follows. Line 9 gives the name of the molecule being represented. Line 10 shows how many atoms (12), bonds (12), substructures (1), features and sets are present within the molecule. 'SMALL' on line 11 states that this molecule is of type 'small molecule' and can be any one of : "SMALL", "BIOPOLYMER", "PROTEIN", "NUCLEIC_ACID" or "SACCHARIDE". Line 12 denotes the type of charges associated with atoms represented later. In the case above, no charges are present, but the line could denote Gasteiger charges or a range of other charge types.

The second record present in the format is the '@<TRIPOS>ATOM'. This record holds information on the atoms present within the molecule. Breaking down the format of line 16, which all other lines up to and including 27 follow, the first number present denotes an identification number for each atom. This is unique and usually increments from a starting ID of 1. The second column is a name for the atom. The following three columns give the x, y and z coordinates of the atom position in orthogonal Cartesian space. The next column gives a Sybyl Sybyl is a commercial molecular modelling package containing many tools and easy to use user interfaces) atom type to the atom, followed by a number denoting the substructure

within the molecule that the atom belongs to. Following this substructure membership column, the name of the substructure is given. Finally, the charge associated with the atom is given. In this case, the "NO_CHARGES" keyword was given in the MOLECULE record, using no charges, and therefore the value here is zero. The next record ("@<TRIPOS>BOND") details bond information. Starting with a unique identifier for each bond, then two columns denoting atom numbers that the bond is to and from. A Sybyl bond type is then given. The last key ("@<TRIPOS>SUBSTRUCTURE") details the substructure to which the atoms belong. The first column of line 42 gives a unique ID to the substructure, the next column a name for the substructure; in this case "BENZENE". In the representation above, column 3 is directly after "BENZENE" and is '1'. This denotes the root atom of the substructure. Column 4 shows the PERM keyword, this can be any one of "TEMP", "PERM", "RESIDUE", "GROUP" or "DOMAIN".

Multiple molecules can be represented in a single mol2 file by simply stating the "@<TRIPOS>MOLECULE" keyword again and continuing the definition of the molecule.

3.2 Dataset generation

When creating datasets or a library of compounds for virtual screening, it is advisable to select carefully what goes into them. As postulated by Walters:

"*Virtual chemistry space contains perhaps* 10^{100} *molecules*". [3]

This number whilst an estimate serves as an indication of how vast chemical space is. Whilst computers allow us to generate millions, perhaps even billions of theoretical chemical structures, there is little use taking the brute-force approach. Walters also points out that:

"*The concept of 'maximally diverse libraries' is not sensible. Focused, synthesisable, drug-like libraries are needed.*" [3]

The above statement affirms two important points. Firstly, the library must be synthesisable. Predictions and simulation could be carried out on compounds that are nearly impossible to synthesise, existing only in a computer and therefore never testable or assayable in a biochemical laboratory. Secondly, the libraries should contain drug-like molecules. This comes from the fact that most drugs share similar properties to other drugs, such as low molecular weight and, typically, they adhere to Lipinski's rule of five [93] (see section 3.3.1 Lipinski's Rule of Five).

The design of a library of compounds which will be used by virtual screening techniques should therefore focus on the quality of the dataset as opposed to the size.

A balance must be struck between two extremes, these being the diversity of the dataset (required for interesting new leads) and the ‘drug-likeness’ of the set.

In a survey carried out in 1996 by Bemis [94] in which a database of 5,120 compounds with shown therapeutic activity were selected, 1,179 frameworks were found, with a large proportion of the drugs contained a common 32 frameworks. These 32 frameworks were identified disregarding atom types and hybridisation states, considering only connectivity. When stricter rules were applied and consideration given to atom types and hybridisation states, 2,506 different frameworks were found in the database of 5,120 compounds. Amongst this set, one quarter of the drugs contained a commonly found set of 42 frameworks. Figure 3.11 shows the top ten most common frameworks found in drugs identified by Bemis.

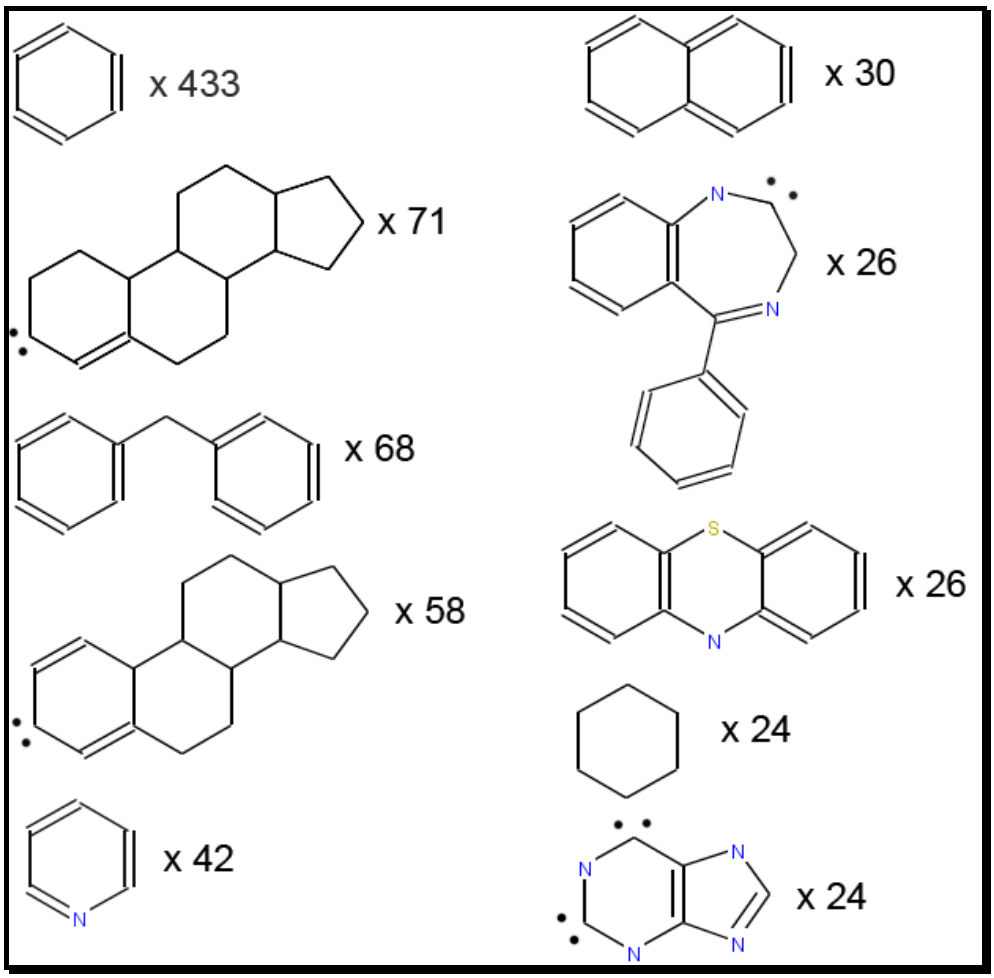


Figure 3.11 - Top 10 most common frameworks found in drugs determined by Bemis in 1996, with occurrences

Later, in 1999, Bemis [95] applied the same technique but this time looking at the side chains found in a database of 5,090 compounds with proven therapeutic activity. Taking into account atom type, hybridisation states and bond orders, 1,246 different side chains were identified. Each molecule contained an average of 4 side chains. Bemis also stated that the diversity that side chains provide is low. Ignoring carbonyls, 15,000 side chain permutations were identified, of these, around 11,000

are from the 'top 20' groupings of side chains [95]. Figure 3.12 shows the six most common of these sidechains.

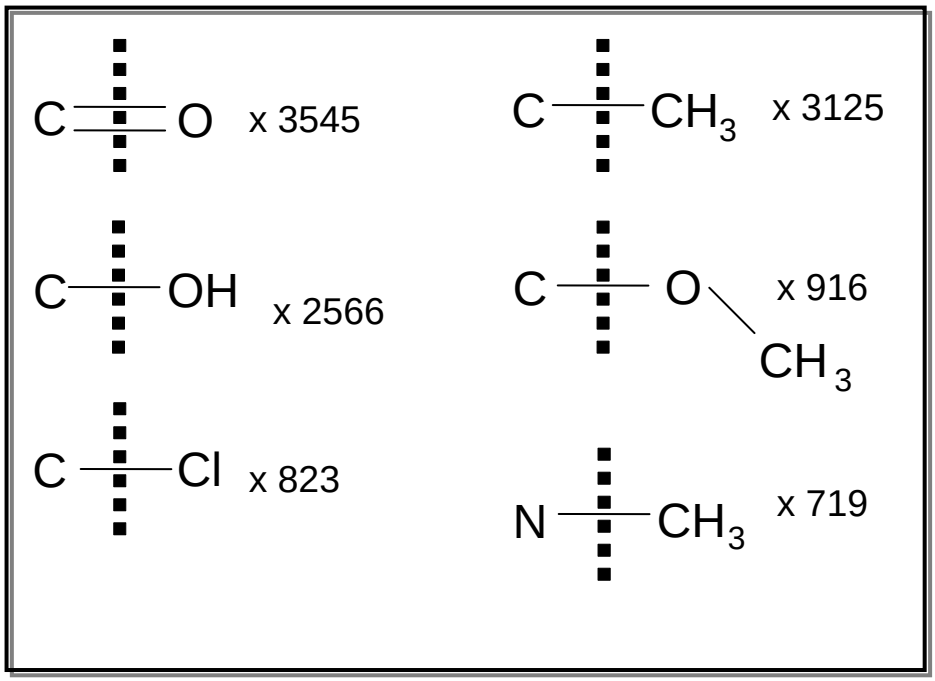


Figure 3.12 - Top 6 most common side chains found in drugs determined by Bemis in 1999, with occurrences. Left of dotted line is the bound scaffold atom, to the right the side chain

There are about 2,500 different scaffolds found in known drugs [94].

It is found that side-chains are attached in ~3 positions; there are around 1,000 different side chains in known drugs [95].

"10,000 scaffolds X (1,000 side-chain groups)^3 = 10^{13} compounds".

[95].

As Bemis states that 10^{13} compounds are of interest, this drastically reduces the number of compounds that should be considered for inclusion in a new virtual screening dataset/library, but it is still at a far too unmanageable level. With vastness

of the drug-like chemical space so large, generation techniques need to be carefully designed and consideration given to the similarity of molecules generated or selected for inclusion. Filtering and similarity methods should be used to ensure that the set is not too similar. As observed by Lyne [96], generating the library could be achieved using similarity/substructure searching [97], pharmacophore matching [98] or 3D shape matching [99; 100; 101].

The resulting library can be cut down by the application of filters, such as Lipinski's rule of five [93], number of rotatable bonds, and polar surface area [3].

3.2.1 The EDULISS dataset

During the course of this research, the standard dataset used for virtual screening and benchmarking is the Edinburgh University Ligand Selection System (EDULISS) dataset. Version 1 of this database created by a previous student, Andrew Hinton, contained 1.67 million small molecules. Recently updated by a co-worker, Kun-Yi Hsin, EDULISS version 2 contains ~3.67 million unique compounds. The selection criterion for molecules being entered into EDULISS is that they are all available from various suppliers. In the context of work carried out within the department, where the testing in biochemical laboratories is of high importance, the high availability of the compounds outweighs other factors such as their diversity and traditional drug-likeness. A web based interface to EDULISS for use by academics is available at: http://eduliss.bch.ed.ac.uk

The website acts as a frontend for the EDULISS database, which represents and stores each compound in the SDF format.

3.3 Dataset filtering

With the availability of a large number of compounds, reduction in dataset size may often be desirable. Presented in the following sections are four criteria which can be used to reduce the number of compounds considered for inclusion in a virtual screening dataset. When virtual screening finds a molecule which is experimentally confirmed to bind, the next stage is often lead optimisation, taking the hit and trying different chemical groups in suitable positions within the molecule. Adding to molecules usually results in an increase in molecular weight and a decrease in solubility. With filters applied to dataset generation, it is possible to pre-emptively select molecules which may be added to and therefore retain their drug likeness throughout their expansion.

3.3.1 Lipinski's Rule of Five

For a potential drug to be of use, it must exhibit good absorption, distribution, metabolism, excretion and toxicity behaviours (ADMET). Lipinski defined a set of simple rules known as 'Lipinski's rule of 5' (RO5) to easily filter out compounds which according to the model would exhibit poor absorption [93]. Applying the RO5 filter to a molecule is easily achieved with a small amount of knowledge about its composition. To pass the R05 and therefore to be considered drug-like, the molecule must pass at least three of the following: number of hydrogen bond donors ≤ 5, number of hydrogen bond acceptors ≤ 10, molecular weight ≤ 500 Da and the calculated CLogP (described in section 3.3.3 Partition coeffieients: CLogP and

MLogP) ≤ 5 (if using MLogP described in section 3.3.3, then MLogP ≤ 4.15). Lipinski calls molecules which pass this test drug-like.

3.3.2 Astex Rule of Three

Similar to Lipinski's Rule of Five, the Rule of Three (RO3) is a filter that, when applied to a dataset, predicts fragment-like molecules [102]. For a molecule to pass the R03, it must pass all of the following: molecular weight ≤ 300 Da, hydrogen bond acceptors ≤ 3, hydrogen bond donors ≤ 3 and CLogP ≤ 3. Fragment like molecules can be useful to indicate functional groups which may be of importance to binding in a given location on a receptor.

3.3.3 Partition coefficients: CLogP and MLogP

Partitioning coefficients represent the ratio of concentrations of a compound in organic and aqueous systems. The most common of these systems is octanol and water and, without the explicit statement of different systems, P will always refer to the octanol water partition ratio. The ratio is often expressed as a log to the base 10 of the coefficient P, giving rise to LogP. Compounds with a LogP > 0 are lipophilic and compounds with a LogP < 0 hydrophilic [103]. LogP is therefore a valuable property which can be used to predict the solubility of compounds. This can be extremely important when designing virtual screening libraries and reviewing potential hits. A strong binder is of little use if it is insoluble at relatively low

concentrations as delivery to the target binding site will be hindered. LogP refers to an experimentally determined value whilst CLogP and MLogP refer to calculated LogP values. ClogP, or calculated LogP, uses a constructionist approach and a library of over 200 fragments, associated with a constant, along with a number of correction factors which can be used to calculate LogP [104; 105]. MLogP denotes a LogP value obtained through the use of the Moriguchi model [106]. MLogP assesses the solvent accessible surface area of a molecule using a probe of a radius of 1.4Å. With this probe 'rolled' over the surface of the molecule, hydrophobic effects of polar groups encountered along with correction factors contribute to the final calculated value [106].

4 Structure based virtual screening and LIDAEUS

This chapter provides some background into virtual screening (structure based) and goes on to describe the operation of the high throughput virtual screening code LIDAEUS. The creation of a parallel framework and subsequent parallelisation of LIDAEUS is then discussed, the parallel version running on a massively parallel computational platform containing 2,048 processors. A publication is available detailing the porting and parallelisation of LIDAEUS [107].

4.1 Virtual screening

Virtual screening is an encompassing term. Here, we are concerned with finding novel non-covalently bound ligands for known protein structures. With proteins responsible for the majority of functions carried out within a cell, the ability to control or turn off proteins by simply blocking their binding pockets creates exciting possibilities. Virtual screening has the same goal of *de novo* ligand design, but attacks the problem of finding inhibitors from a different angle. Virtual screening is used to test a database of ligands against a target. *De novo* ligand design constructs ligands that should bind to a target based on either structural data or an ensemble representation of the target derived from known binders [7; 8]. Virtual screening is commonly seen as the first stage of the drug discovery process and is commonly referred to as *in-silico screening*. Many different codes and approaches to this simulation exist and vary greatly in their computational complexity. With enough computing power it is possible to simulate protein ligand binding very accurately using quantum mechanical derived approaches [4; 5; 6]. However, this report is concerned with what is commonly known as high throughput virtual screening, using simplified methods to achieve higher throughput [108]. This term encompasses codes and approaches that test in the order of thousands of protein-ligand complexes in one run.

Typically, the virtual screening process involves representing the binding pocket of a protein and then the execution of two steps often merged together or used to direct

each other. Potential ligands are normally extracted from a database of available compounds. Filters can be imposed on the potential ligands to ensure they meet a certain criteria or are not too similar [109]. The ligand is positioned in the binding pocket. This is called posing the ligand and the process may generate multiple poses (fitting many different ways into the binding pocket). Each pose is then evaluated by a scoring function. The line between the two may be blurred by the scoring function helping to direct the initial pose generation, or minimisation by the scoring function drastically changing the pose at the scoring stage. With ligands able to generate multiple poses, the computational complexity bears little relation to the number of ligands being tested. The success of virtual screening is prevalent in the current literature. An overview of recent successful discoveries of novel protein-ligand complexes is given by Taylor [110]. The conceptual steps in virtual screening are shown in Figure 4.1.

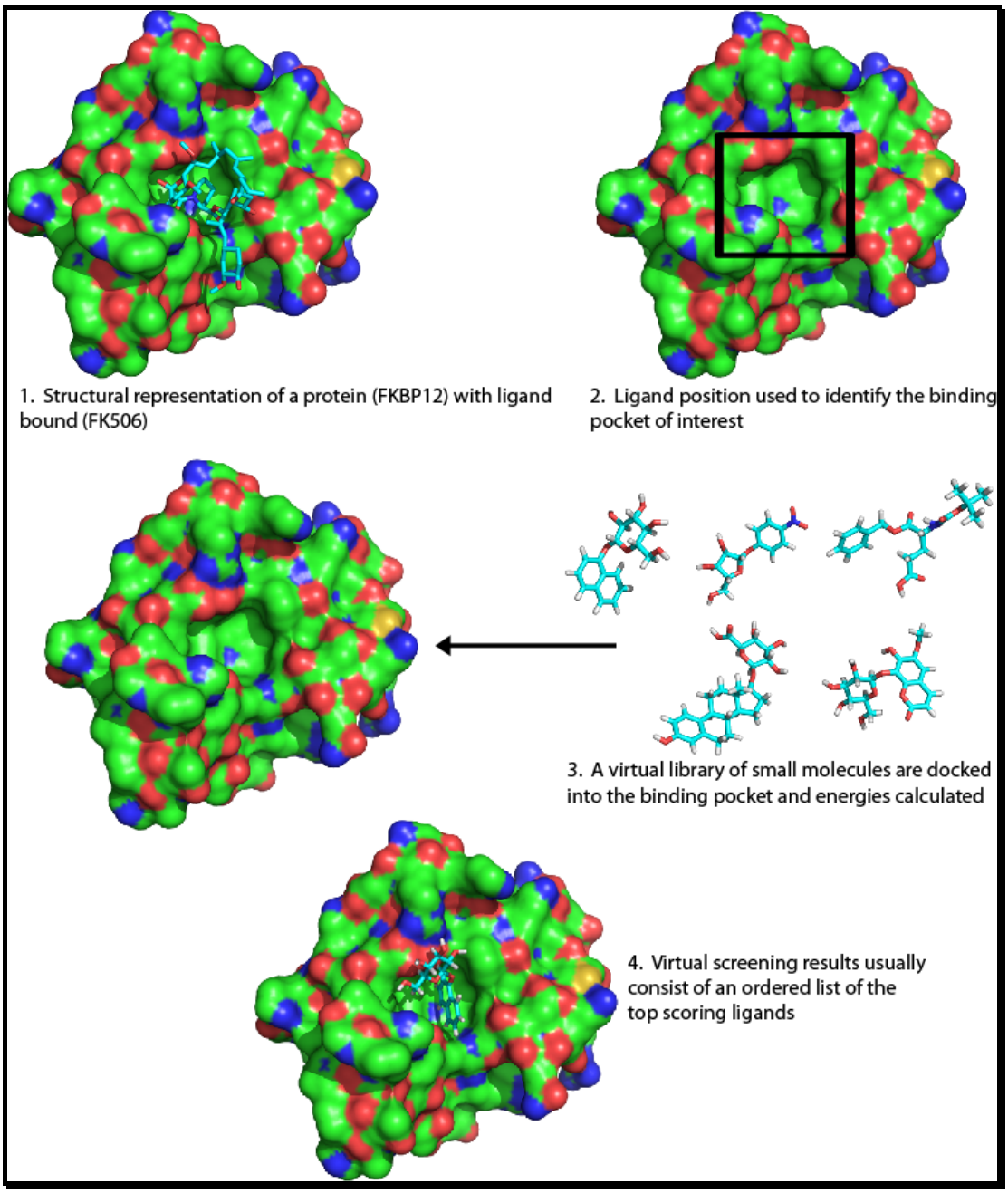


Figure 4.1 - Conceptual steps in structure based virtual screening. Colour scheme of protein surface as follows; carbon is green, nitrogen blue and oxygen red. Colour scheme of ligands as follows; carbon is cyan, oxygen red, nitrogen blue.

Depending on results obtained from the screening of thousands of potential ligands and further investigation, it may be worth experimentally testing the binding properties of these ligands in the laboratory.

A number of problems present in protein-ligand virtual screening have been identified in studies to include:

> *"difficulties in treating accurately the effects of solvation, the entropic changes due to hydrophobic interactions and conformational changes, and receptor flexibility."* [111]

These difficulties and others can all be categorised into three problem classes in the field of protein-ligand virtual screening, each being addressed differently by available virtual screening codes:

1 Ligand positioning – How should we explore the position/orientation of ligands during docking? Ligand flexibility, small molecules are not rigid and may exist in many conformations, do we need to test them all and how should they be generated? [112]

2 Scoring functions – How can we extract a single score to indicate the binding affinity between a protein-ligand complex? [113]

3 Target flexibility – Protein binding pockets are not rigid. The conformation may change during binding. Side chains may move, affecting binding energies. [114]

Another small consideration is that most codes do not consider the possibility of ligands bonding covalently to the protein. There are certainly codes that address this problem using branches of quantum mechanics, but these can in no way be considered high throughput. Some high throughput codes such as FlexX can be directed to bind the ligand covalently but this requires specific knowledge of the complex and is not discovered automatically [115]. Use of high throughput virtual screening codes in this way requires knowledge of the complex, and since the focus of this report is on discovery, the use of this kind of codes is not considered.

4.1.1 Ligand Positioning

Bringing a small molecule into complex with a protein appears to be a trivial task. As stated previously, a standard step is to use a natural ligand to define the binding site/pocket. Once the area where the ligand is to be docked has been identified, the candidate ligands can be positioned within it. With every area of space explored by translation of the small molecule, rotation comes into play. It may be a good strategy to explore 1 degree increments along each axis. With 1 degree increments along each axis, 360^3 or 46,656,000 orientations must be tried for a full rotational search. Ideally, this search will be carried out at increments within the space of the binding

site. This leads to the observation that an intelligent search must be carried out, drastically reducing the number of positions/evaluations that need to be carried out. Four commonly found search algorithms are detailed below. A detailed analysis of docking methods are available in reviews by Kellenberger [116], Kitchen [117], Schneider [13] and Taylor [118].

Monte Carlo methods

Monte Carlo [119] refers to a class of algorithms that use random numbers to aid in their computation and progression towards a result. Virtual screening programs such as AutoDock [120; 121; 122] and MCDOCK [123] use Monte Carlo methods, generating random numbers to direct translation/rotation of ligands and accepting/rejecting the changes based on improvements obtained in predicted binding energies.

Fragment based

The virtual screening programs FlexX [124] and DOCK [125] use a fragment based approach. Breaking down the potential ligand into smaller parts, a base fragment is chosen and docked by itself into the binding pocket (using a variety of methods). The ligand is then constructed step by step with direction given by the predicted binding energies.

Genetic algorithms

Virtual screening programs such as GOLD [126] and AUTODOCK [120; 121; 122] use genetic algorithms to aid in ligand positioning. Genetic algorithms essentially apply the idea of evolution and survival of the fittest to a problem, allowing viable solutions from a pool of potentials to evolve and improve over time. In a simple example, the position of a ligand may be represented as a chromosome. A pool of these solutions is generated and a fitness function used to decide which solutions go on to evolve through mutation and crossovers with other 'fit' solutions. After many generations, it is hoped that surviving solutions will be viable ligand positions.

Systematic

A virtual screening program docking ligands exhaustively in every possible position at a predefined resolution could be defined as systematic. This systematic search is a search through space; however, a systematic search could be applied to something that itself represents a search through space but limits the combinatorial potential and therefore the size of such a search. An example of this is in the virtual screening code LIDAEUS [107; 110; 127], which identifies favourable positions for ligand atoms to overlay and uses combinations of these points and ligand atoms to rotate/translate the entire ligand into a position that overlays a predefined number of these points.

4.1.2 Scoring methods/functions

Scoring methods aim to put a quantitative value on the binding energy of a protein-ligand complex and fall under one of three categories which may be merged together. The categories are force field, empirical and knowledge based. A detailed analysis of scoring functions is available in reviews by Ferrara [128], Stahl [113], Taylor [118] Wang [129] and Xing [130].

4.1.2.1 Force field based scoring functions

Force field based methods of scoring are the most common approach of quantifying the interactions between a protein and a ligand. A list of parameters allows the summation of energy contributions from van der Waals and electrostatic interactions. Due to the scope of force field base scoring approaches, difficulty lies in including the displacement of solvent, and so scoring of this nature considers the complex to be in vacuum.

4.1.2.2 Empirical scoring functions

Empirical scoring as implemented by methods such as Ludi [131; 132] and Chemscore [133] uses regression techniques to fit experimentally determined binding energies to the complex of interest. This fitting of data using linear regression relies upon a training set. With a model created from the training set, it can be applied to the complex being studied, and expands on the abilities of force field methods by including entropic factors into the calculation.

4.1.2.3 Knowledge based scoring functions

Using a repository of data from experimentally determined structures, atom pairing preferences and frequencies as well as atom environments are examined. Comparing predicted binding modes of protein-ligand complexes to real world data, features matching experimentally confirmed observations are scored highly which contribute to the final score of the predicted complex. PMF [134] (Potentials of mean force) and DrugScore [135; 136] are implementations of knowledge based scoring functions.

4.1.2.4 Consensus scoring

As the name implies, consensus scoring uses a variety of scoring methods and combines results in order to give an indication as to the strength of binding in a protein-ligand complex [137; 138; 139]. Studies have shown that the combination of scoring functions, often weighted with coefficients, increases the accuracy of the predicted binding strength [113; 138]

4.1.3 Target flexibility

Virtual screening approaches have often treated the receptor as a rigid body, considering the ligand as flexible during protein-ligand docking. Few treat the receptor as flexible. This comes down to a matter of computational complexity. Simulating a flexible receptor is extremely expensive, and full flexibility is often achieved through the use of molecular dynamics programs such as GROMACS

(Groningen Machine for Chemical Simulations) [58] and NAMD (Nanoscale Molecular Dynamics) [61]. The virtual screening code AutoDock addresses the problem by allowing partial flexibility to be defined within the receptor. The importance of receptor flexibility is shown in literature, such as the discovery of a number of HIV integrase inhibitors by Schames [140]. Here, molecular dynamics were used to investigate receptor flexibility and uncovered a binding site that was previously inaccessible.

4.2 LIDAEUS

LIDAEUS (Ligand discovery at Edinburgh University) [107; 110; 127] is a high throughput virtual screening program written by Dr Paul Taylor of the Institute of Structural and Molecular Biology, University of Edinburgh. LIDAEUS is a highly adaptable and modular rigid body docking virtual screening code written in C++. In its original state it existed as a series of programs which communicate through UNIX pipes.

The execution of a standard LIDAEUS run takes the form shown in Figure 4.2**.**

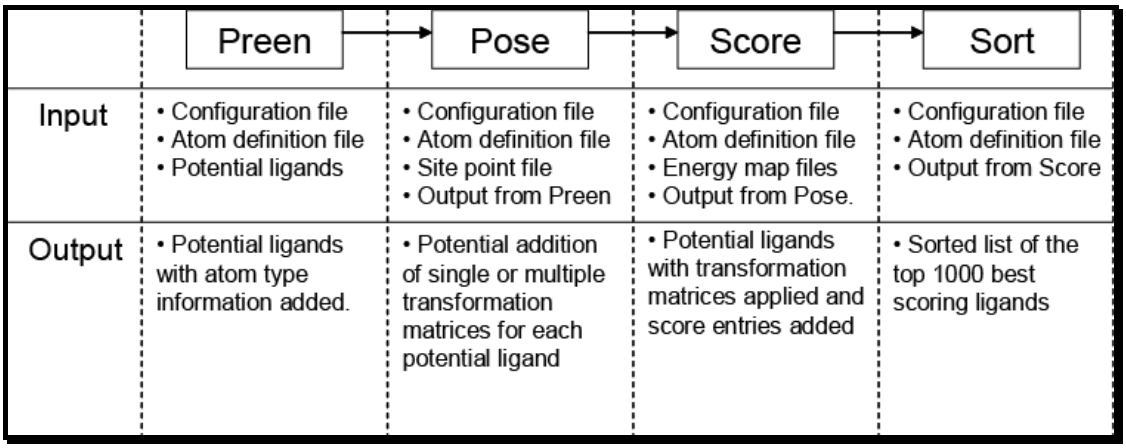


Figure 4.2 - The LIDAEUS pipeline

LIDAEUS requires a small amount of job preparation before a run; this takes place once to screen as many potential ligands as required. The following example of a virtual screening run uses the PDB file with ID 1FKJ representing the protein FKBP12, a protein of interest in immunosuppressive therapies. The first stage of a run is to obtain (normally from the Protein Data Bank [15]) a PDB file of the protein in complex with its natural ligand. From this, two structural representations are created.

The first being the protein alone, excluding the ligand and the second the ligand in the same frame of reference as the protein such that if the two files were merged, then the ligand would appear in the correct position in complex with the protein. A number of programs can be used to achieve this, the most convenient being PyMol [43]. The protein is required to be in the PDB file format and the ligand in the Tripos mol2 file format [141]. Using these two files, the next stage in the job preparation is to generate the map files and the site point file. Map files represent a volume within which at periodic intervals along each of the three dimensional axis, energy values are defined.

A word on endianness:

Microprocessors can be grouped into the choice of endianness they employ, the options being big-endian and little-endian. Endianness refers to the byte order used by microprocessors. The concept can be illustrated using the English language and addition. If writing or reading the number 1073, the first digit is the most significant, the last digit the least. Reading and writing numbers in the English language is big-endian. It may appear that this big-endian representation is the best and most logical number storage solution but when carrying out most simple mathematical operations, people use little-endian notation. The simplest way for a person to add 1073 and 1259 on paper or mentally is to first carry out the addition of the least significant digits first (3 + 9). Then work backwards carrying any larger units backwards towards more significant digits. Many microprocessors differ on their byte order for reasons such as this, an addition of two integers may require working on the least

significant bits first and so little-endian order is preferred although considered by a person to be the least intuitive representation of numbers.

The naming convention big-endian and little-endian is in reference to the book Gulliver's Travels by Jonathan Swift.

> *"Swift's hero, Gulliver, is shipwrecked and washed ashore on Lilliput, whose six-inch inhabitants are required by law to break their eggs only at the little ends. Of course, all those citizens who habitually break their eggs at the big ends are angered by the proclamation. Civil war breaks out between the Little Endians and the Big Endians, resulting in the Big Endians taking refuge on a nearby island, the kingdom of Blefuscu. The controversy is ethically and politically important for the Lilliputians. In fact, Swift has 11,000 Lilliputian rebels die of the egg question"* [142]

A map file generation program is then used to generate three maps. The code for this is an adaptation of methods originally implemented by Professor Malcolm Walkinshaw whilst working for Sandoz on the program Probis and is similar to the methods used by the GRID program [143] to identify protein binding sites. Map files are generated and written to binary files in big-endian format. LIDAEUS was originally run on big-endian Silicon Graphics, Inc (SGI) machines and so the

convention for map files has remained the same. This does not cause a problem on little-endian CPUs as long as map files generated on such platforms are converted with a tool to big-endian representation before being run. By default, LIDAEUS expects big-endian byte ordering in all map files regardless of the current platform. The structure of a map file takes the following form: First 12 bytes contain 3 floating point values denoting the x, y, z coordinates of the origin of the volume in the frame of reference prescribed by the protein in the PDB file. A protein may be given to the map file generation program, with the protein translated an amount along any axis, therefore altering the x,y,z position of the origin of the volume with respect to the space referenced by the protein. The next 3 floats (occupying 12 bytes) denote the grid spacing between points for the x,y and z axis. Commonly, this is 0.5 Å for each. From the origin of the volume in the 3D frame of reference dictated by the protein, information at a later position in the map file will describe energy at intervals of 0.5 Å along each axis in 3D. Following this, the next 12 bytes represent 3 integers. These are the number of values along each axis. Following this, 4 byte floats appear in the map file and make up most of the remainder. With the previous information, the number of floats required to populate the volume with energy values can be calculated; intervals on x axis multiplied by intervals on y axis multiplied by intervals on z axis. Following this large amount of information is 6 more floats occupying the last 12 bytes of the file. These represent the unit cell associated with the map. However, these values are present for legacy compatibility and are not used by LIDAEUS to any useful degree as the assumption is made that conventional orthogonal 3D space is being operated in.

Generating the values stored in the framework of a map file is carried out using the program mapsite. The program generates three required map files for a LIDAEUS run. The three maps describe van der Waals, hydrophobic and hydrogen bonding energies [127]. The van der Waals (vdW) map is defined as follows by the classic Lennard Jones potential:

$$E(\text{vdw}) = \sum_{n}^{1} (A / r(gp)^{12} - B / r(gp)^{6}$$

The sum is over set n where 1-n are all atoms. r is the separation between the protein atom and the probe. A, B and gp are coefficients from the Amber molecular dynamics package [144]. The hydrogen-bond donor and acceptor energy maps are defined in a similar way as the van der Waals map but include a weighting term dependent on the deviation from ideal hydrogen bond angles. This weighting is encoded by high energy values in the map file, which can be visualised as a halo of ideal hydrogen bonding angles when contoured at a suitable level.

Stepping through the volume occupied by the protein, at regular intervals, an interaction energy is calculated using definable probe atoms. For the default van der Waals as described above, a simple water molecule or uncharged oxygen is suitable. Complimentary probes are chosen for the hydrogen bond donor and acceptor maps and energy calculations, but this time, the force field constants that are used to quantify the energy have an additional term. As hydrogen bonding occurs most favourably when the atoms are joined at a specific angle, the weighting applied is a

cosine of the ideal angle minus the actual angle present. This creates the halo of energy described above.

Mapsite takes as input environmental variables dictating where to find the protein as a PDB file, the ligand as a mol2 file, what the names of the newly created map files should be and the filename of the site point file to be generated. Command line arguments also dictate weighting factors that can be applied to the energy calculations and the amount of extra space that should be included in the map files beyond the bounds defined by the protein. The three maps are a volumetric representation of specific energies associated with the space surrounding the protein.

Using these three maps, the energy experienced by a ligand in any position can be determined by the energy contributions from each map on each atom. Figure 4.3 shows a surface generated from the van der Waals map where the energy is zero and is visualised using the molecular viewer and modelling package WITNOTP [145].

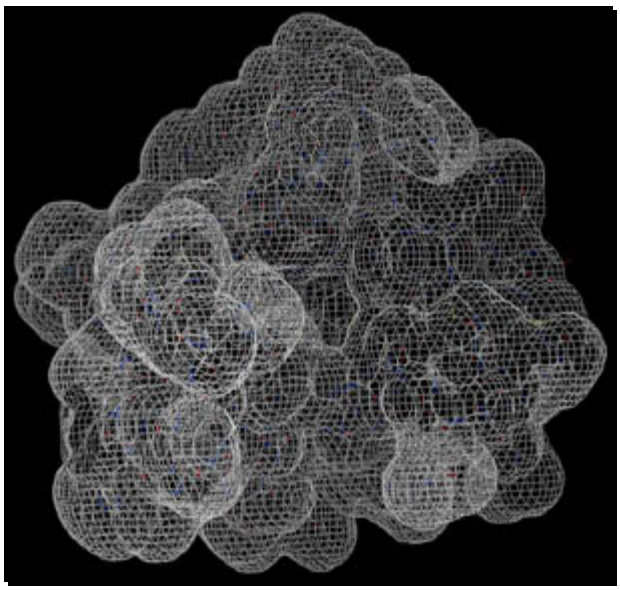


Figure 4.3 - van der Waals map of FKBP12 isosurface at zero energy

As proteins may contain multiple binding sites, the site point file directs LIDAEUS to dock potential ligands at a certain location. When supplied the ligand, Mapsite will use the space defined by a bounding box around the ligand to identify the area of interest on the protein. Within this area, energetically favourable points for different types of atoms will be identified. This may for example indicate that a hydrogen bond acceptor can favourably be placed in this position. The default number of site points which Mapsite generates is normally 170 although it is user definable. The generated site point file is output in the mol2 file format. This is then converted to a file of type CLU (a custom ASCII file format) using the program mol2clu, which takes as command line arguments the location of the site points in mol2 file format. The output can then be redirected by the user to a file with a '.clu' extension. Shown in

Figure 4.4 are the site points generated from FKBP-12 overlaying a PyMOL representation of the protein [43]. At this stage, all of the information required to carry out the run has been generated.

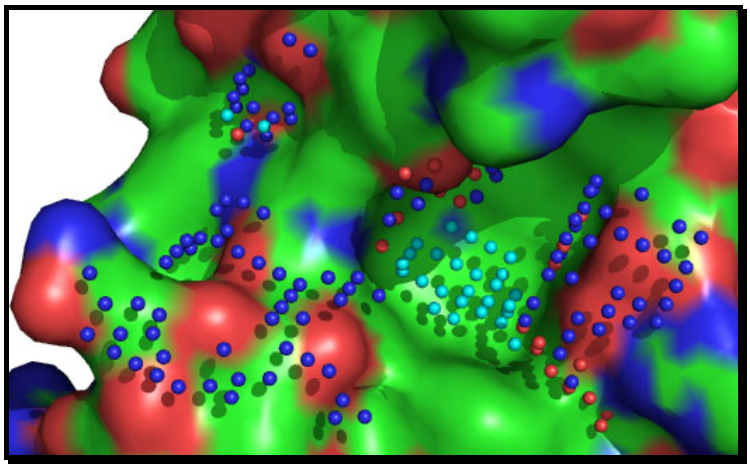


Figure 4.4 - Site points guide docking to FKBP12

4.2.1 Preen

As the pipeline is composed of multiple programs reading from standard input and output, potential ligands can be supplied from a variety of sources. Typically, an SDF file containing multiple entries is piped to preen. All programs in the pipeline consider one molecule at a time before passing it on to the next program. Preen reads input in the SDF file format. For any use to be made of the molecule, the atoms must be "typed", this process adds more information such as hybridisation states to the atom. Typing is done using an extensive set of rules and is aided by recursive ring finding routines that traverse the molecule's connection table to a depth of eight atoms looking for ring structures. Extended atom type information is added to an internal representation of the molecule using the TRIPOS forcefield atom types. These types give greater information on the hybridisation and geometry of the atoms; hybridisation denotes characteristics of an atom that vary according to the bonding interactions and the state of those bound atoms.

4.2.2 Pose

After information has been added to the molecule by preen, pose tries to fit the molecule to the set of site points in as many ways as possible simple example shown in Figure 4.5.

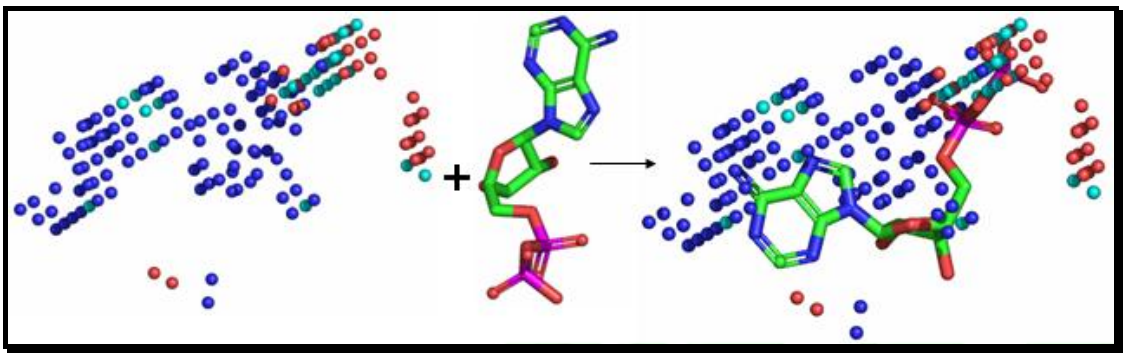


Figure 4.5 - Small molecule overlaying sitepoints

This is the most computationally expensive part of the pipeline. The pose module takes its name from the action of positioning (i.e. “posing”) a ligand against a target. Many poses may be generated by a potential ligand. Because LIDAEUS is a rigid-body docking code, suitable poses for the ligand are achieved through rotation and translation. Instead of iterating through all orientations and positions of the ligand at a coarse resolution, the pre-generated site points are used to constrain the positioning search. LIDAEUS generates a list of distances between bound atoms in the ligand as well as a list of distances between each site point (up to an adjustable cutoff value). A search tree is used in the following set of actions to explore combinations of fitting points. Each stage of depth within the tree matches ligand atoms to site points with increasing tolerance values. These values are user-definable, but by default, the first four atoms are fitted with tolerances 0.02Å, 0.04Å, 0.06Å and 0.08Å. The list of distances between bound atoms in the ligand is checked against distances between site points. If a match is found, then a test is carried out to see if the next bound atom in the tree fits a distance present between nearby site points. If the depth of the tree of fitting atoms reaches a user-definable depth (by default four), then the ligand

in the correct position is able to satisfy the atom type requirements described by the site points. Once it is known that a set of certain ligand atoms can overlay a set of site points, the 3D coordinates are extracted and LU-factorisation used to obtain a combined rotation/translation matrix that overlay the key ligand atoms onto matching site points. This matrix is then stored with the representation of the ligand. Applying this matrix to the 3D coordinates of the ligand produces a pose. With ligands able to generate multiple poses, the computational complexity of the problem bears little relation to the number of ligands being tested.

4.2.3 Score

Molecules arriving at the score program have information on atom types, and a list of matrices applicable to the coordinates of each atom, orientating the ligand to overlay site points. With the molecule orientated into a potential binding position, the energy maps are used to sum the contributions being made on each atom. The resulting score has units of kcal mol^{-1}. A six-dimensional conjugate gradient energy minimisation is then applied to the molecule as directed by the predicted binding score in an attempt to refine the previously generated pose. Subsequent translations and rotations are applied to the molecule's position, but this time taking no account of the site points. It is worth noting that transformations are applied to all atoms (representing a rigid body). The energy experienced by the molecule from the map files is used to place the molecule in a more energetically favourable position. This process is repeated for each pose entry belonging to a molecule. As the coordinates

for the molecule in each of its poses will have changed, SDF entries are generated for each pose. The SDFs contain the same molecule but with different coordinates, posing matrix (representing the position before minimisation) and an individual score line showing contributions from van der Waals, hydrophobic, hydrogen bonding energies and the sum of these energies.

4.2.4 Sort

Sort is simple in its operation, with parameters read in from a configuration file requesting (for example) that the top 1000 ligands be kept. Sort simply reads in the scored SDF entries, maintaining a list of the best 1000 top scoring ligands. The program understands that scoring mechanisms may have been used apart from the standard LIDAEUS score module. These other scoring functions may have been placed in the pipeline for additional data discovery or as a sanity check. Sort can calculate the sorting score as a linear combination of values extracted from the comment fields in the mol entry. At the end of a run, this is output to a file for analysis. Visual inspection is a simple method of evaluating the results and the resultant complexes can be visualised with a variety of different tools. Shown in

Figure 4.6 is the predicted complex made by a small molecule and the FKBP12 protein.

Many different factors affect the range of scores achieved in the final sorted list, but a good scoring ligand (therefore predicted to be a strong binder) has a score less than about -20 kcal/mol. Score can be made more complex than this however, allowing the user to up or down weight interactions such as hydrogen bonding. The flexibility of all of LIDAEUS allows complete user customisation and the standard three map files need not be used at all; with the proper definitions a user may define their own maps and scores accordingly.

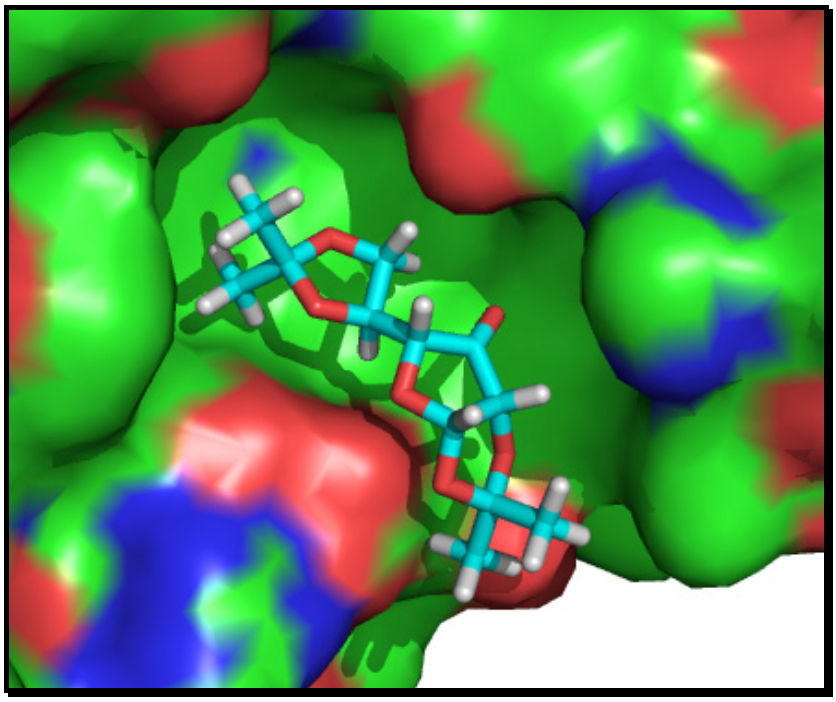


Figure 4.6 - LIDAEUS predicted complex between FKBP12 and a small molecule

4.2.5 Cross platform stability

Any numerically intensive code may be prone to producing inconsistent output across a range of platforms (See chapter 2). During the initial investigation into LIDAEUS, it was discovered that the rigid body minimisation that takes place in the score module was producing inconsistent results across platforms. The inconsistencies were traced to the conjugate gradient minimiser used to slightly alter the ligand's position according to the energy landscape represented by the generated map files. To explain the source of the inconsistencies, the concept of a minimiser must be explained. The simplest example would be that of an energy landscape which creates a contour similar to that of a one dimensional valley. If a marble was rolled into the valley, then it would come to rest near the middle, in a position where the marble was as low as possible. Here, gravity has performed the job of a minimiser. The implementation of a steepest descent minimiser would generate the lowest energy solution for the marble in the valley problem, taking the current location of the marble and exploring the energy in both of the possible directions, left and right. If there is a lower energy position on the left, then the marble is moved to the left. This continues until the marble is in an optimum or near optimum position. If we increase the number of dimensions present in the energy landscape, then it could be thought of as a marble rolling down a 3D valley. If the valley has a clear channel running through it then the marble could easily be positioned in the lowest energy location by a steepest descent minimiser. However, a marble in a valley may 'zigzag' many times to the left and then right sides of the channel. To stop this zigzag effect, which is harmful as many unnecessary movements are made, a conjugate gradient minimisation method can be applied. Here, the choice of new

positions for the marble is influenced by the previous direction from which the marble moved, as well as the new suggested position. As this combination of vectors in space is used, a zigzag effect will be eliminated after a few movements. The minimisation routine in LIDAEUS is conjugate gradient.

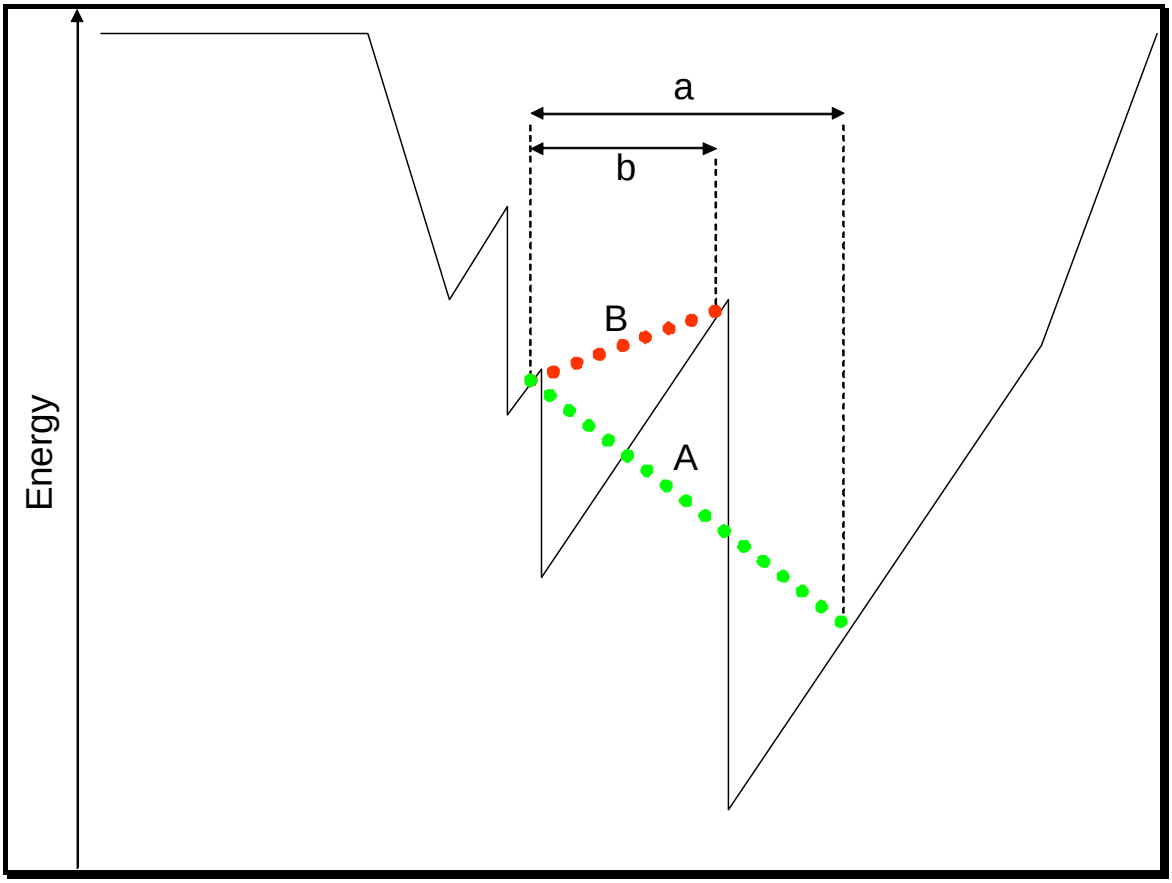


Figure 4.7 - A theoretical 1D energy landscape. Using measurement b, a non favourable positive energy gradient is found (B - red). Using measurement a, a negative and desirable energy gradient is found (A- green)

Inconsistencies were seen when running LIDAEUS on chips of different architecture. The problem was found to be in the sampling size used by the minimiser.

Calculating an energy gradient in one direction often results in a very noisy (jagged) landscape. Calculating the gradient of a move in a certain direction was found not to be using a large enough interval. This is demonstrated in Figure 4.7: using a small interval for calculating the gradient has resulted in a positive energy gradient whilst using a larger interval lead to the discovery of a negative and favourable move. Increasing the interval size used by the conjugate gradient minimiser found in LIDAEUS, results appear correct across platforms and the amount of noise present in the minimisation process has been reduced.

4.3 Parallelisation of LIDAEUS

4.3.1 Motivation

A standard dataset for benchmarking purposes exists and contains 1.67 million small molecules (in the SDF format) that may be purchased from 24 catalogues obtained from suppliers. In this benchmark, the receptor target for these molecules is known as FKBP12, which plays an important role in immunosuppressive therapies. The structure is taken from the Protein Data Bank and has the PDB identifier 1FKJ. This dataset is so large as to be unmanageable in a serial or low-processor-count parallel-processing environment. Selection of a random set of 30,000 molecules, and subsequent simplistic predictive scaling calculations, show that this dataset used with its target would take 29 days to analyze on an AMD Opteron 246 Linux system with 2GB RAM. Runtime lengths such as this restrict users to the screening of small datasets in an effort to obtain the results they need in the amount of time available to

them. HPC systems are an attractive platform for VS runs. The ultimate goal in the parallelisation of LIDAEUS is reduction in runtime for the standard dataset, therefore allowing researchers more freedom to experiment and set up even more complex runs with larger datasets.

4.3.2 BlueGene/l

The University of Edinburgh's IBM e-Server Bluegene/l has delivered high performance computing resources to its researchers. With terascale performance in the range of 4 TeraFLOPS ($4x10^{12}$ Floating point operations per second), researchers have experienced a significant increase in available computing resources.

Although the resulting code is designed to be portable and run on any large scale HPC platform, the target architecture for this work has been this Bluegene system. A detailed description of the architecture and hardware is available in a review by Chiu [146].

Similar to the machine QCDOC [147], a purpose built machine for quantum chromodynamics, Bluegene takes the form of a massively parallel distributed memory machine (MIMD architecture). Bluegene uses many simple and low powered (both energy and performance wise) 700MHz processors and couples them together through a custom interconnect to facilitate fast communication. The result is a large number of simple, inexpensive low powered processors connected via a

custom-designed high performance interconnect. One node (also known as a chip) contains two Power440 cores, with associated caches running at a low clock frequency (700MHz). Low levels of heat dissipation allow the processors to be packed closely together in a small package, allowing one Bluegene/l cabinet to contain 2,048 processors. Each processor core has access to 256Mbytes of memory. Due to the unique nature of Bluegene, the parallelisation had to adapt well to being run on many relatively slow processors, and in a small memory footprint of 256Mbytes.

4.3.3 Parallel implementation and analysis - Bluegene

The original version of LIDAEUS ran on a seven-node openMosix cluster consisting of seven AMD XP 2600 processors, each with one gigabyte of RAM and with a standard gigabit Ethernet network acting as the interconnect. The system therefore consisted of a cluster of standard “desktop” machines managed by openMosix to create a cluster capable of distributed computing. This functionality was achieved by process load balancing between nodes (machines). By starting multiple processes and pipelines of LIDAEUS on each node of the cluster and running the molecular data through a simple multiplexer program, which distributed molecules to instances of the pipeline, we allowed the operating system to migrate pipelines from nodes with a high workload to nodes under a lesser load.

While this approach to parallelisation enabled more results to be gathered in a reduced amount of time, the need for a truly parallel version of the code stems from the restrictiveness of the openMosix parallelisation and its ability to only run in an openMosix environment. Running the code on modern HPC systems would allow the use of a large number of processors that is much greater than the seven used in the cluster.

Parallelisation of LIDAEUS for the BG/L system at the University of Edinburgh takes the form of an MPI "task farm" that uses data decomposition as the parallelisation strategy. Essentially, one processor runs a master process, reading in the small molecules to be complexed with the protein and then serving them to waiting worker processes residing on other processors. The worker processes run their own preen, pose, and score code. As with any parallel code, computational load balance must be addressed. With the intended use of the code being the screening of more than 1 million molecules in one run, they can be served to worker processes in relatively small serving sizes (around 5 to 50 molecules at a time). This greatly limits the scope for load imbalance.

The initial parallelisation was to be specifically for describing the binding pockets of a range of proteins. A co-worker in the department, Yi-Gong Sheng, applied a selection criteria to the full RCBS Protein Data Bank (as of March 9, 2005), selecting crystallographically determined structures with a resolution better than 1.7 Å that

made at least four hydrogen-bonding contacts with the protein and no covalent bonds between the protein and ligand. Sequence identity searching was then used to obtain a diverse set of proteins with a sequence similarity not exceeding 90%. This specific searching problem did not require sorting or retention of only the top results, because information on the best-scoring minimised and non-minimised pose for each molecule was retained. The code has been used to screen a collection of 1.67 million molecules against a resultant selection of 387 protein-binding pockets. The results of this large computational run are stored in a database, and mining techniques are used to investigate the effectiveness of molecular descriptors in predicting binding affinities. Figure 4.8 shows that the lack of a parallel sort, and any lack of any interprocess communication, results in almost linear scaling and a parallel efficiency up to 512 processors. On 1,024 processors, parallel efficiency for one receptor is as low as 54%.

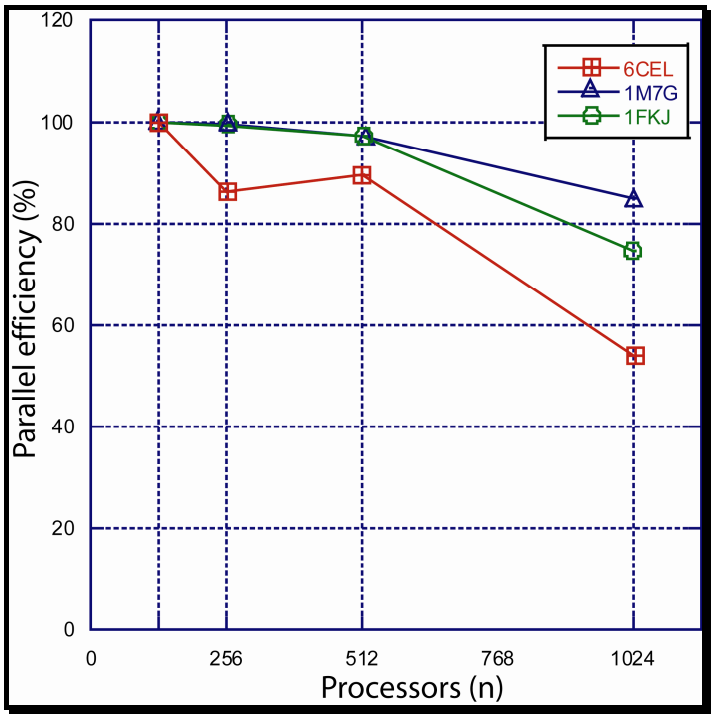


Figure 4.8 - Parallel – LIDAEUS, no sort. Parallel efficiency on a range of processors (128 as the baseline) for the standard dataset against three different target receptors, PDB ID's shown

Further investigation is required, although a possible explanation for low efficiency is that, for 1,023 worker processes, each writing their results to a file at unpredictable intervals (with the master process reading from disk), the network switch through which the BG/L file I/O travels becomes saturated. While the total amount of data written in a run of the standard dataset is manageable at approximately 15 gigabytes, the constant writing of a small amount of data by 1,023 streams may explain the poor parallel efficiency. The scaling data used to produce Figure 4.8 suggests diminishing returns beyond 1,024 processors. A version of the code discussed later removes the requirement for such large amounts of IO and subsequently shows improved scaling.

As previously stated, site points offer a way to constrain the search space in the positioning of ligands. A more exhaustive search of this space can be carried out by alteration of a user-specifiable variable called “resol”. This variable acts as a tolerance value that denotes the distance from which the centre point of key ligand atoms must be from site points. The default value of 0.02 Å may be increased to carry out a more exhaustive search. As this value increases, the computational complexity exhibits exponential growth. Massive numbers of VS runs, dealing with millions of small molecules, are not suitable for larger tolerance values. Because the goal of VS is the discovery of interesting drug leads, higher tolerance values are useful only when using VS runs on smaller targeted subsets or classes of small molecules. However, this tolerance value offers the ability to alter the amount of computation required to process a small molecule. With this in mind, along with the consideration that the limiting factor (I/O) will remain constant with increasing number of poses, it is possible to set up runs with larger number of poses that scale to and beyond the limit of a single BG/L cabinet utilizing 2,048 processor cores.

In order for the parallel version of LIDAEUS to be used as an HTVS tool in the rigorous sense of the term HTVS, we did not collect information on the best poses for each molecule but instead, implemented a sorting algorithm to produce a reduced list of the top-scoring results from all poses made by all molecules. An interesting characteristic of the BG/L architecture, specifically the current version of its Compute Node Kernel (CNK), is that no exceptions or errors are thrown when a chip

has filled its local RAM and a stack overflow event occurs. The lightweight nature of the CNK means that there is also no virtual memory. Depending on the mode in which each node is run (i.e. CO or VN mode), each chip has either 512 MB or 256 MB of local memory respectively. With the possibility of a ligand binding to a protein in a variety of poses, one small molecule may produce many positions and scores that need to be retained or at least considered for retention at the sorting stage. In a restricted-memory environment such as this, a problem arises. We are attempting to sort a dataset so large that all of it cannot be held in one memory location. The sort cannot proceed on one processor. A collaborative effort amongst processors must be made.

Any time that more complexity is added to a parallel code, consideration must be given to its effect on the overall execution time brought about by the new code interaction with existing components. In addition to the two existing classes of processor, the master, and the worker, we add a new class; the sink. Conceptually, the sink sits at the bottom of the task farm into which the workers dump their results. If the workers simply performed the screening and then indiscriminately communicated their results to the sink, the performance of the code would decrease due to the communications overhead. It is therefore important to reduce the amount of communications generated. To achieve this objective, we introduce the idea of a cut-off value. At the start of the screening process, all worker processes perform the screening on their data, filling up a small (sorted) local buffer. When this buffer is full (typically when it contains about 50 poses), the contents are sent to the sink

which, depending on their scores, adds them to its sorted list. When the requested sort size has been achieved by the sink (e.g. a size corresponding to 1,000 molecules), a cut-off value can be calculated. A new pose that does not meet the requirements of the cut-off can be disregarded as it will never enter the sorted list. This cut-off value is then communicated back to the workers so that this new criterion can be applied to their local buffers. A conceptual overview of the taskfarm is given in Figure 4.9

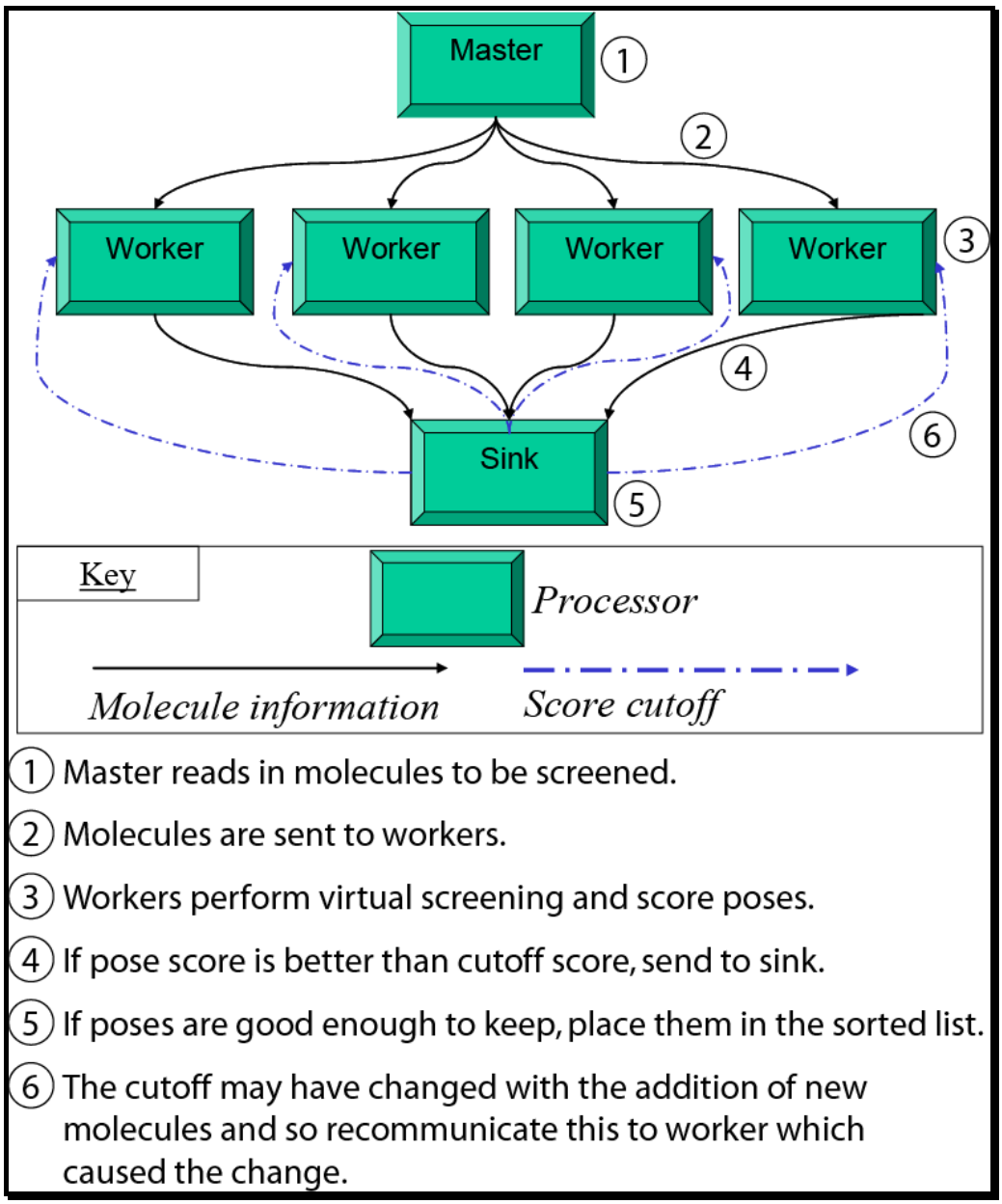


Figure 4.9 - Conceptual flow of information during a parallel LIDAEUS run

Consideration must be given as to when this cut-off value should be communicated. One approach is to allow the worker processes to communicate their buffers to the sink and then request a new cut-off value (assuming that their data changed the cut-off). In another approach, when the worker initiates communication with the sink, a cut-off is received. The worker then removes entries from its buffer that do not meet the criteria and proceeds with the send. We chose the first pattern of suggested communication. At the start of the screening process, with the cut-off value set high to allow potentially all poses to be communicated to the sink, the code has a performance decrease. As the screening progresses, the cut-off is updated and communication to the sink is reduced. A logical approach to reducing this performance decrease would seem to be to set the cut-off to allow only very good dockings to enter the sink list. This approach may be undesirable because when a sorted list of 1000 of the best scoring poses is requested but less than 1000 poses make it past the cut-off, the list will not contain 1000 elements. A counterintuitive result discussed later shows that when other variables, such as the worker buffer size, are set sensibly, almost the reverse should be applied to ensure shorter runtimes. Aside from the number of processors used for the VS run, three user-tuneable variables exist that relate to the parallel sort, that is, the previously mentioned score cut-off starting value, the worker buffer size, and the number of elements the final sorted list should include. A failure to specify values results in the use of predefined defaults. Interestingly, the algorithm can be described as self-tuning. As long as the worker buffer size is not set to an unsuitably small or large value (e.g. around 1 or greater than 1000), the score cut-off value, which dictates communication, converges to a suitable value no matter how large its original starting value. As a consequence,

small runs can be set up to explore the behaviour of the program when a larger proportion of its runtime is spent with an unoptimised score cut-off. Figure 4.10 represents a standard scan of 1.67 million compounds against the receptor target FKBP12.

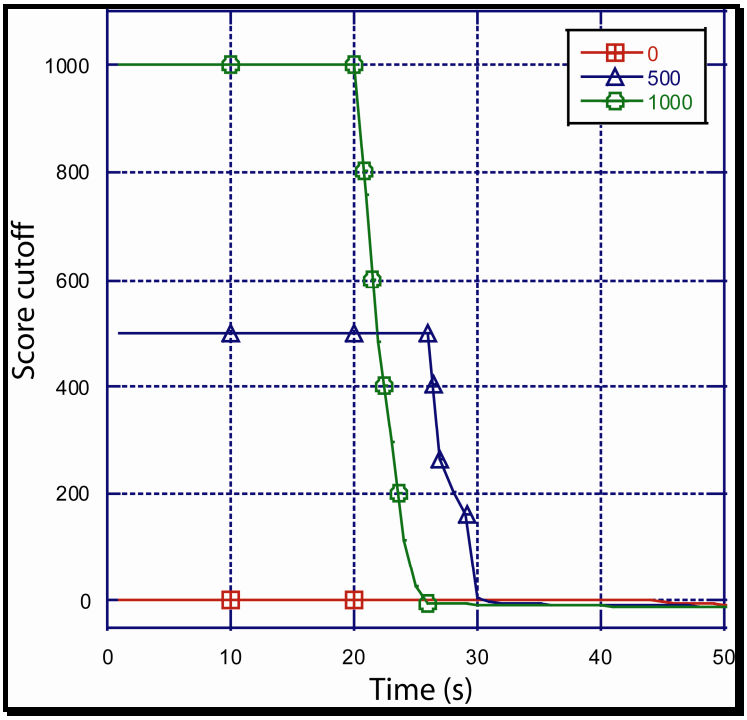


Figure 4.10 -From the code implementing the parallel sort, convergence of the sinks score cutoff value from three different initial values of score cutoff. Collected using the standard dataset against receptor represented in PDB ID 1FKJ

When the score cut-off is started at three drastically different values (0, 500 and 1000), convergence of the score cut-off values occurs extremely quickly. It seems counterintuitive that the larger the score cut-off value is set, the slightly shorter the overall execution time. We observed runtimes of 10,983, 10,977, and 10,973 seconds respectively using default settings and three different starting-score cut-off values, running on 128 nodes (256 processor cores). The speed of cut-off

convergence can be seen in Figure 4.10, with the value for each scan at or below 0 within 31 seconds of program execution. At this point in time, the lowest established cut-off is achieved by the scan using an initial value of 1000. This observation for runs with a higher starting score cut-off is due to the fact that the contents of local buffers will be communicated many more times than those for runs with a lower cut-off. This higher rate of communication is caused by more molecules making it past the cut-off and filling the buffers earlier. As the scan with the higher starting-score cut-off progresses, the communicated buffers will typically contain a range of results varying greatly in their score. For example, some results are just able to enter the buffer through the score cut-off, while others easily enter the buffer. In another run with a more restrictive cut-off, the local buffers can be said to contain higher-quality results but are communicated less frequently. Essentially, results of poorer quality are being used to flush the good results in the local buffers through to the sink. A lack of communication keeps the sink cut-off values artificially unrestrictive. This flushing, and the ideal rate at which it should occur, is an artifact of the previously mentioned user-definable sorting variables. Figure 4.11 shows runtimes for the standard dataset against a receptor with PDB identifier 1FKJ. 256 processors are used for a range of worker buffer sizes and initial-score cut-offs. It is important to note that this version of the code, which is implementing the parallel sort, is working on a different problem than the original problem with no sort.

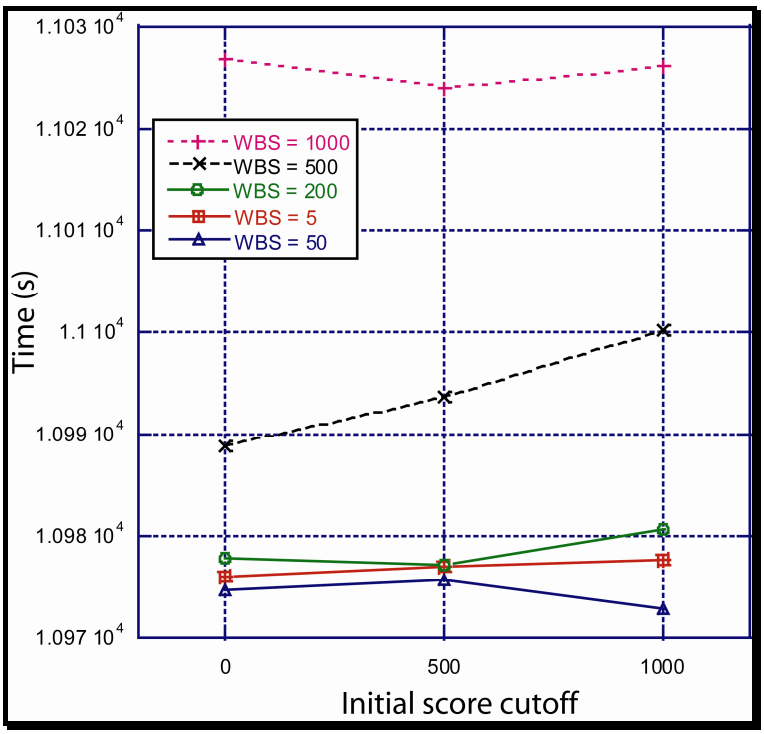


Figure 4.11 - Execution time of the standard dataset complexing with structure from PDB ID 1FKJ on 256 processors. A range of worker buffer sizes and initial score cut-offs are shown.

The problem for which the initial code version was used concerned the profiling of binding pockets. Here, the best minimised and non-minimised poses for a molecule were written to disk. This is different from the action of the code when carrying out a true VS run where the best of all poses are to be kept or sorted. Here, every pose must be considered for output to the latter stage of sorting. One molecule has the potential to generate many poses. Thus, instead of considering just two poses per molecule, multiple poses are considered. This is reflected in profiling results: the time taken for the same dataset in both versions of the code typically takes 1.5 to 1.7 times longer when each pose generated is considered for retention. Essentially, the two codes are incomparable. As can be seen from Figure 4.12, profiling the code implementing the parallel sort has resulted in almost 100% parallel efficiency up to

1,024 processors. It is worth noting that profiling shows parallel efficiency at over 100% in some instances. This level of efficiency is due to the fact that the value is calculated using 128 processors as the baseline, and with more processors, more worker buffers are being flushed, filling the larger sink buffer and establishing a cut-off earlier. Cache effects are a common cause of super linear speedup (parallel efficiency of over 100%); however, this is not the case here.

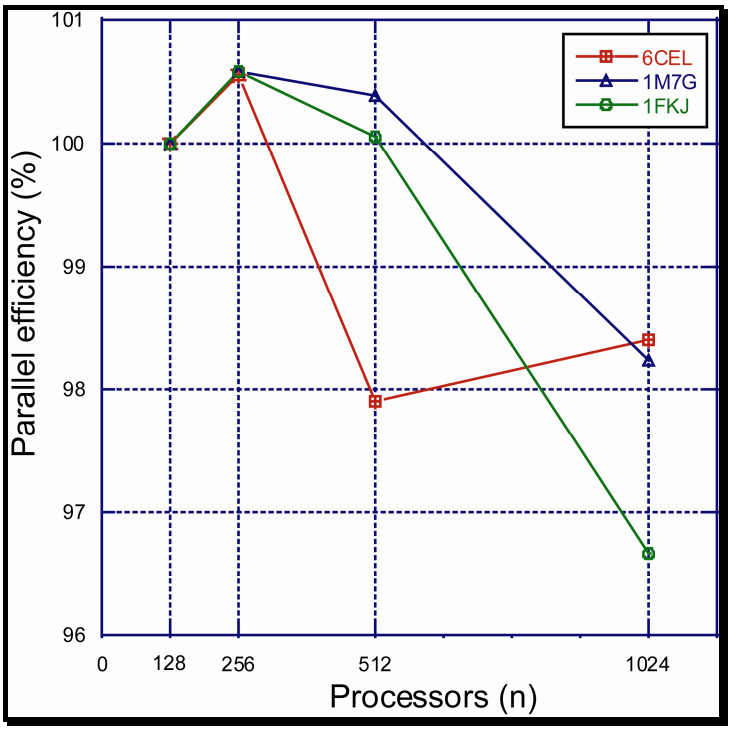


Figure 4.12 - Parallel – LIDAEUS, with sort. Parallel efficiency on a range of processors (128 as the baseline) for the standard dataset against three different target receptors, PDB ID's shown

This result supports the previous hypotheses that the code without the parallel sort is being I/O bound at higher numbers of processors. The self-optimising nature of the sort has ensured that most of the computation, after early convergence of the score cut-off, proceeds in almost a trivially parallel manner.

Development has tailored the code to the lightweight massively parallel characteristics of the BG/L architecture. The main goal of LIDAEUS has not been departed from, that is, LIDAEUS is a tool for drug lead discovery and not for predicting the exact binding mode of a ligand at the expense of increased and more restrictive computational runtimes.

4.3.4 Parallel implementation and analysis – Commodity hardware (Opus)

During the later stages of the project, the department acquired two eight core machines. The opportunity was present to port and profile the parallel implementation of LIDAEUS to this hardware. Each of the two machines contained two quad core Intel Xeon E5420 processors clocked at 2.5GHz, giving each machine 8 CPU cores and 8 gigabytes of RAM. With the intention of the parallel LIDAEUS implementation being that it was highly portable (written using MPI to facilitate the parallelisation), an opportunity existed to see how flexible the implementation was. The machines were set up with a private gigabit Ethernet network between them to facilitate message passing only. This private network would ensure that messages did not have to go through the standard network hardware used by the department which may be under heavy load and therefore cause delays in the delivery of MPI messages. A queuing system was also set up on the machines using openPBS (Portable Batch System) and Torque [148; 149]. Configuration of openPBS allowed MPI programs to treat the two 8 core machines and one 16 core machine, between which messages could be passed allowing the two machines to work together or collaborate on the same job. From this point on, the cluster made by the two 8 core machines, will be referred to as Opus. Opus uses an open source implementation of MPI called openMPI [150]. Alongside this implementation are wrappers for the GNU Compiler Collection, allowing the compilation of MPI programs. The step of porting the code to Opus was virtually non-existent, simply calling the openMPI compiler wrappers produced functioning executables.

In profiling the parallel LIDAEUS on Opus, it must be remembered that the amount of processing power available is severely limited when compared to Bluegene. For this reason, profiling could not be performed using the previously used dataset of 1.67 million compounds. The code is profiled on a range of processors, using a dataset of 226,364 compounds from the supplier Specs and the protein FKBP12 (PDB ID 1FKJ). As the implementation uses two processes running on two cores for management of the parallelisation (master and sink), running on three cores means that only one worker is present. With this taken into consideration, running with a range of workers and calculating parallel efficiency on worker numbers only gives around 100% parallel efficiency all the way up to 14 workers. With 16 cores available on Opus, it would be beneficial to allow cores that are running master and worker processes to participate in carrying out work, enabling 16 cores to work on the virtual screening run. However, when the live usage of each core was examined, it was evident that both cores running master and sink code were always at 100% load. Investigation into this phenomenon revealed that both the master and sink code made use of calls to MPI_Waitany. Calling this puts the process into a loop constantly checking if any MPI messages have been received, if nothing has, then it immediately checks again. Locked in this loop, the core in which this process is being run has no time to do anything else but check and recheck message status until a message is received. This was not ideal as essentially the two cores could not be used for any other computation. To solve this problem, the calls were replaced with the following code:

```
int testanystatus=false;

MPI_Testany(size-2, requests, &dest,&testanystatus, &status);

      while(!testanystatus){

      usleep(sleeptimemaster);

      MPI_Testany(size-2, requests, &dest,&testanystatus, &status);

      }
```

This code makes use of MPI_Testany which checks once to see if a message has been received. Whilst a message has not been received, the following is carried out:

- Sleep for a specified amount of time
- Test if a message has been received

The key component of this change is the call to sleep allowing the master or sink process running on the core to stop execution and yield to other processes present. The change detailed above allows a virtual screening run to be carried out on Opus using 16 cores and spawning 18 processes. The master and sink hardly use any processing time and so can be located on any of the other cores, sharing execution time with another worker process. This change to the parallel implementation has been successful but required a small amount of profiling to optimise the amount of time the master and the sink processes should sleep for. If they sleep for too long, this could mean that the worker processes are left waiting for work. Sleep for too short an amount of time and a high proportion of CPU time is spent checking if a message has arrived. It was found that around 400 microseconds was optimum.

However, it is expected that this optimum value would decrease in an environment where a higher number of cores were available.

With the above modifications, parallel LIDAEUS was run with 18 processes spawned (1 master, 1 sink and 16 workers) on the 16 available processors, this leads to the standard dataset as used for benchmarking on Bluegene (FKBP12 from PDB ID 1FKJ and the original EDULISS dataset containing 1.67 million compounds) running to completion in 9 hours 18 minutes and 47 seconds.

Had the parallel code remained on Bluegene where the high availability of processors meant that the master and sink could happily reside on their own cores, then there would have been no need to remove the 100% CPU usage phenomenon from the master and the sink.

4.4 Live visualisation of the LIDAEUS process

A program has been developed to allow user-directed LIDAEUS virtual screening. The program is called LidVis and is written in C++ and makes extensive use of the LIDAEUS code. The code does not make use of any parallel computing features and is instead intended to be a way for users to manually dock ligands to a protein target. With guidance from the user, the ligand can be brought into complex with the protein using 3D rendering and visualisation techniques. A score or binding energy can then be calculated for the complex using standard LIDAEUS map files. The ability to pose ligands using LIDAEUS site point matching and user directed positioning exists. The minimiser can be run at any stage of the process, applying small movements to the ligand to achieve better scores. This can act as a refinement to the human guided positioning. Once a complex has been created, the ligand and its positioning information as well as LIDAEUS score keys can be written out to a file (in the SDF file format). Loading the SDF output into other programs along with the original protein PDB file used for complex creation allows for re-creation of the complex and therefore further actions can be carried out using common tools.

The program is designed to be run on modern Linux systems. Testing and development was carried out on a dual core Intel Centrino system running Ubuntu 5.04 Linux. The program uses the Mesa graphics library [151] and openGL [152; 153] for 3D graphics and acceleration. The program is controlled via the keyboard , manipulating the 3D environment.

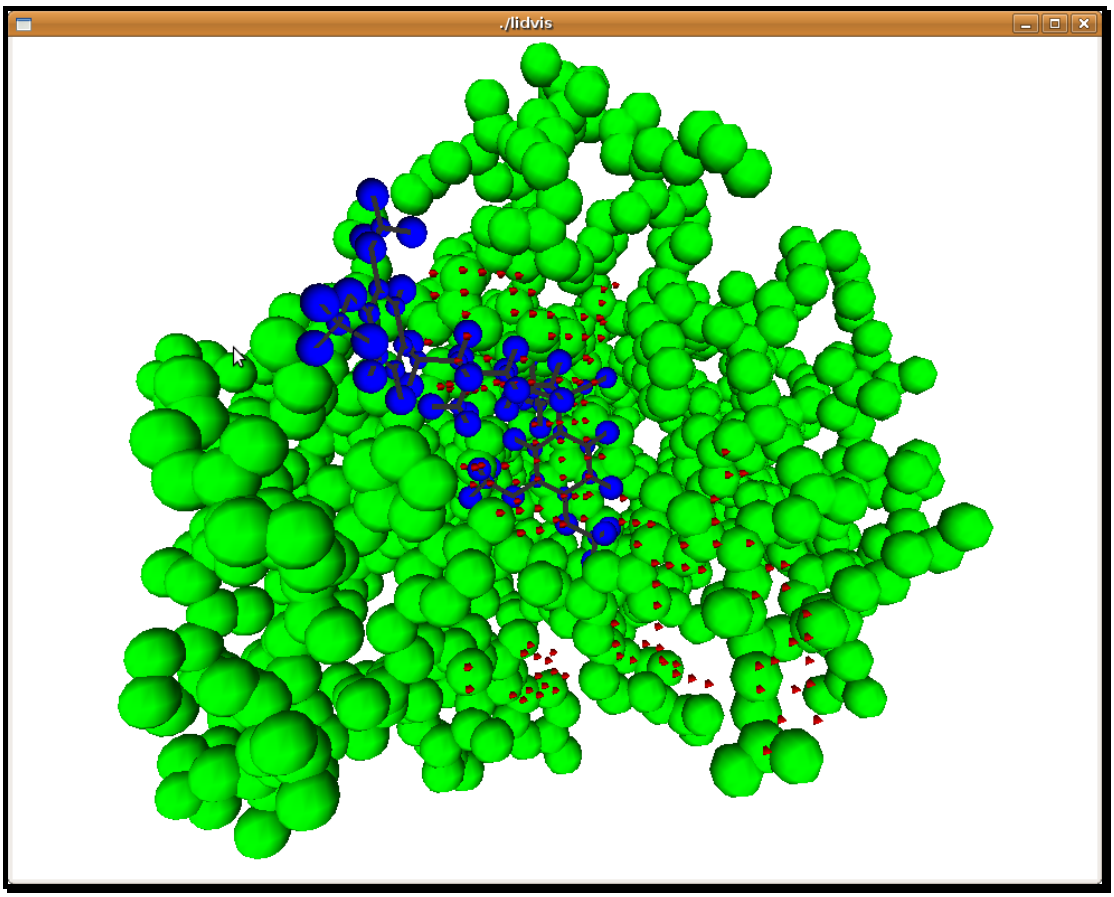


Figure 4.13 - Visualisation of a predicted protein-ligand complex using LidVis. Green spheres represent the carbon alphas of protein residues, blue spheres ligand atoms and red triangles LIDAEUS sitepoints.

Figure 4.13 shows a LidVis session where the user has manually positioned a small molecule and brought it into complex with the protein. In this case, the protein is FKBP12 from the file with Protein Data Bank identifier 2DG3. Scores for the complex can be written out as well as a representation of the ligand in its current position. The rendering of the complex within the 3D environment takes the form of solid green spheres representing residue/amino acid positioning, smaller blue spheres represent the ligand atoms which are the same size as the corresponding atom's van der Waals radii. LIDAEUS site points are represented as the small red spheres.

There being no ability to distinguish between site point types, they act more as a guide and navigation aid when the user is looking for the binding pocket of the protein.

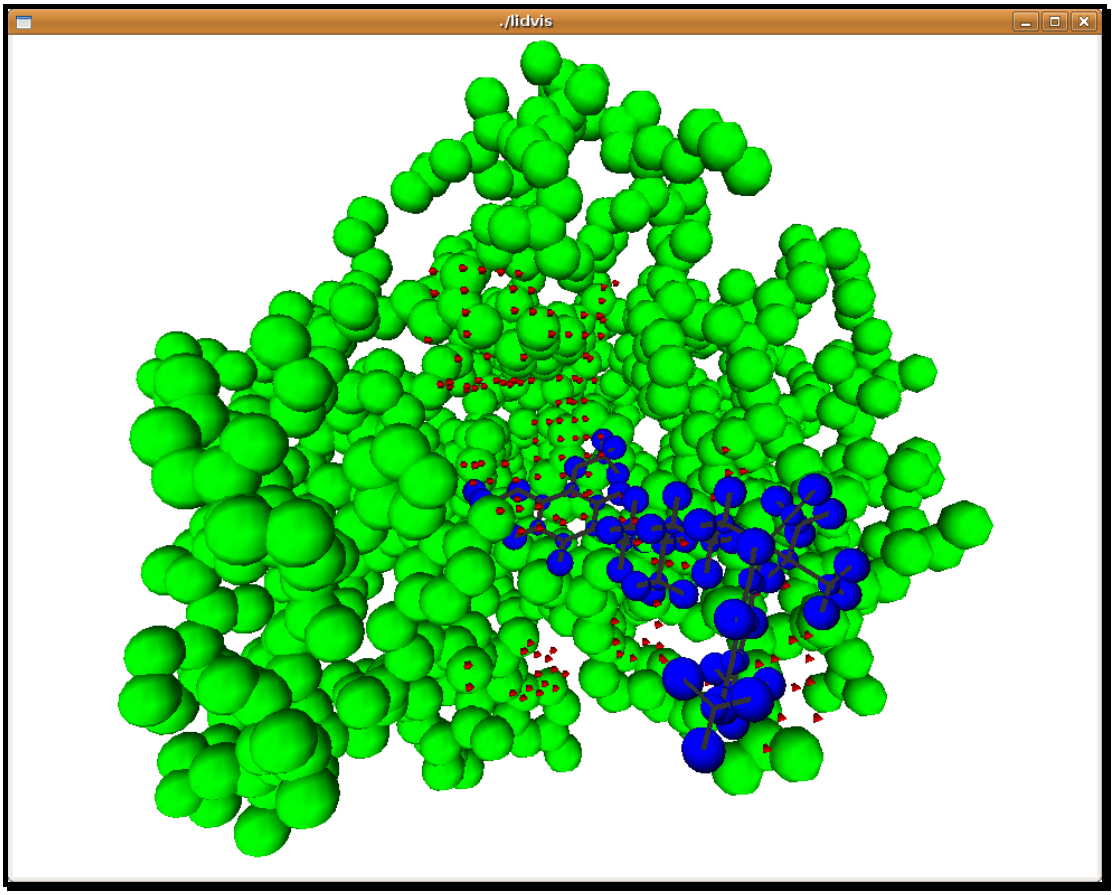


Figure 4.14 - Visualisation of a predicted protein-ligand complex using LidVis

Figure 4.14 shows a similar situation with the same protein and ligand found in Figure 4.13. This time, the LIDAEUS posing system has been used to position the ligand and the top scoring pose is shown.

4.5 Simple flexible docking in LIDAEUS

A drawback of LIDAEUS is that it is a rigid body docking program. The success of a ligand creating an accurate complex with the target protein depends greatly on the input conformation that the ligand is supplied in. This is not too much of a problem with small molecules as these will typically not have too many low energy conformations. In the case of larger potential ligands, perhaps peptides, the performance of LIDAEUS would be expected to be less than ideal. Any large flexible molecules will cause problems as the conformational space cannot be explored. To address this problem, an attempt at implementing flexible ligand docking within LIDAEUS was tried. It was a goal that the addition of this flexible ligand binding ability should not too greatly impact upon the time taken to try complexes. This is an objective statement, but overall, LIDAEUS should stay as a high throughput virtual screening program.

A strategy was devised. Ligands would proceed through the LIDAEUS pipeline in the standard way, but an augmented score routine would enable ligand flexibility. LIDAEUS would pose the molecule in the regular way, apply rigid body minimisation techniques to achieve the lowest possible energy for the pose and then apply the force field to the ligand, the protein remaining rigid. A simple steepest descent minimiser was written which works on each of the three dimensions of every atom. The minimiser, with the ability to move ligand atoms (making the ligands flexible) evaluates the energy experienced by the molecule from both intramolecular

forces (from the force field) and intermolecular forces (from the LIDAEUS maps representing the protein).

For flexibility to be implemented, a force field was created. This force field was kept intentionally simple, taking only four terms. The force field applied to molecules has terms and the ability to calculate energy contributions from bond lengths, bond angles, torsions and van der Waals interactions.

The force field borrows heavily from the simple empirical force field documented by Leach [80] and relies on the empirical data contained within the Sybyl 6.91 parameter set [154].

Figure 4.15 shows the form taken by the force fields energy calculation routines.

$$e = \sum_{\text{bonds}} \frac{1}{2}k_b(l - l_0)^2 + \sum_{\text{angles}} \frac{1}{2}k_a(\theta - \theta_0)^2 + \sum_{\text{torsions}} \frac{1}{2}V_n\left[1 + \cos(\text{nw} - \gamma)\right] + \sum_{j=1}^{N-1}\sum_{i=j+1}^{N}\left[4\epsilon_{i,j}\left[\sigma_{\text{ij}}{}^{6}\right]\right]$$

Figure 4.15 - The form of the simple force field

l_0 denotes the reference length of a bond between the specific atom types. The current length is given by l and a constant k_b gives the stiffness of the bond; a measure of how much energy it takes to change the length of the bond from its reference value. The expression for calculating the bond angle energies is very similar, this time the difference in radians from a reference value being acted upon by another specific stiffness constant. The expression for torsional energies is slightly more complex as typically, torsions have an element of periodicity to them. The number of minima is factored into the equation by the symbol n, w being the location of the first periodic minimum. The last term sums the van der Waals energy experienced by the molecule. Each atom interacting with every other atom is included in the calculation. From the parameterisation of the force field, values for the van der Waals radii are known and from this, the energy between two atoms in space can be derived. Experimental code was written to investigate the effect of only calculating van der Waals interaction energies for atoms in close proximity to each other. Due to pipelining effects on modern processors, the added conditional in the code to check two atoms proximity slowed execution and global interaction energy was found to be quicker to calculate.

The force field described above was implemented in C++ so as to allow easy integration at a later stage with LIDAEUS. Tests carried out on simple small molecules such as decane showed a sensible predicted conformation. Figure 4.16 shows a decane molecule in vacuum making the transition from a high energy planar conformation to its well known spaced out conformation. Output from the program

shows that using the simple force field and minimiser, the decane molecule went from a starting intramolecular energy of 14.97 kcal/mol to 6.64 kcal/mol. The minimiser also performed well on a buckminsterfullerene or buckyball.

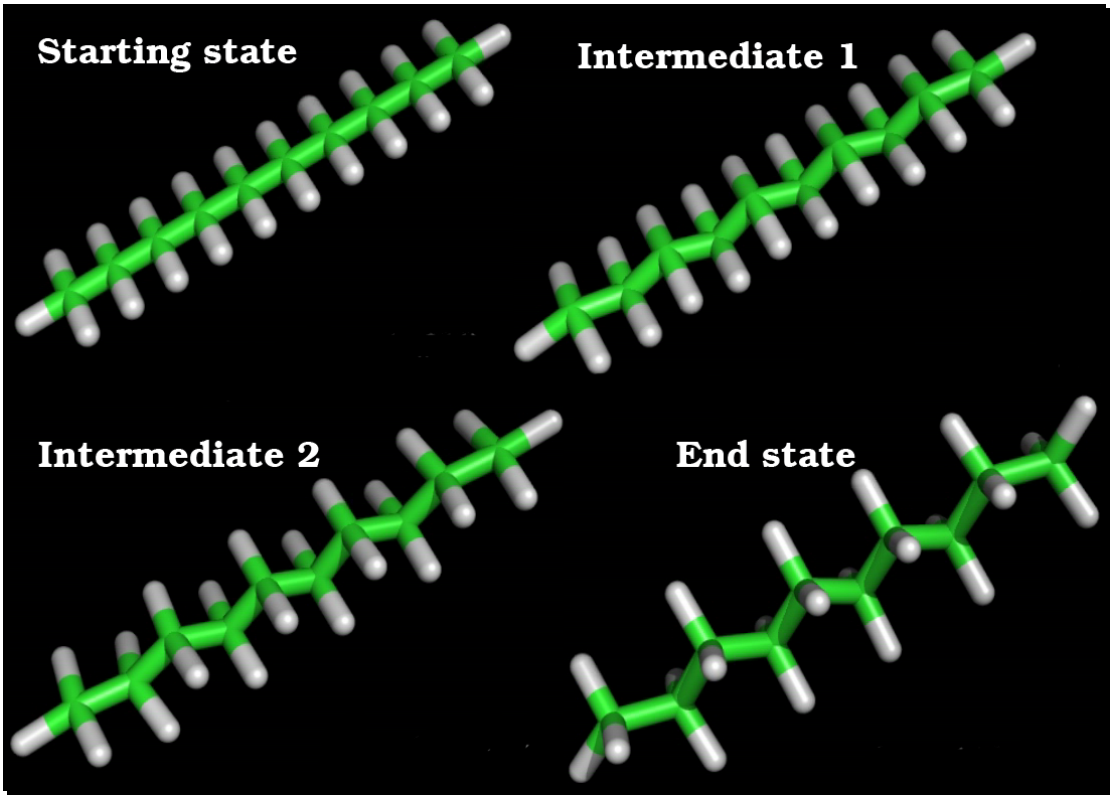


Figure 4.16 - The simple minimiser working on a high energy planar conformation of decane

Adapting and applying the simple force field and steepest descent minimiser to the LIDAEUS scoring routines provided mixed results. Often poses were not suitable for flexible minimisation and when they were, the conformations would be erratic and obviously incorrect. For example, benzene rings were minimised to non planar conformations, so any structure with a ring in it would likely be minimised to a

completely incorrect conformation. Carrying out the flexible docking was also slow, taking two minutes on modest hardware to reach a minimal energy conformation of the molecule shown in Figure 4.17. Investigation into the non planar ring phenomenon revealed a number of additional terms that would need to be added to the energy function in order to keep ring structures planar, such as cross terms and non-bonded interactions which lead to slightly different ideal torsion angles and help to keep the appropriate rings planar. Expansion of the force field terms in this way was deemed unnecessary for a simple investigation into ligand flexibility. A simple approach was adopted in order to keep rings planar. The force field code was expanded to chunk the molecule into a set of islands. An island can be defined as a ring structure and its hydrogen atoms. If two ring structures are joined together, then this forms a larger island. These islands can be rotated and translated by the minimiser. With the ability to roughly minimise ring structures based on intermolecular forces, the minimiser was expanded to take into account the energy contribution on the molecule from the energy maps of the protein binding pocket. Conformational changes were directed by both the intermolecular and intramolecular forces. These methods were employed on the poses generated by the original score code. This means that the rigid conformation is posed multiple times into the binding pocket, each of these poses goes through rigid body minimisation techniques and finally each atom/island is minimised using the force field with contributions from the LIDAEUS map files.

Initial inspection of newly predicted high scoring conformations appears to follow the surface of the protein well and position ring structures well, however, it is evident

that, for the force field and minimiser to work well and mimic real world results, more terms need to be added, increasing the complexity and runtime. Figure 4.17 shows a flexible run undertaken with a molecule binding to FKBP12. The runtime for this one calculation on the single starting pose supplied was over two minutes. This is unacceptable as the advantage LIDAEUS has over other virtual screening codes is the speed in which complexes are simulated. For this reason, it was decided that no further work would be carried out to improve the flexible ligand binding addition to LIDAEUS. It is available as an optional module which could, in the future, see use with small datasets. Typical LIDAEUS runs will continue to simulate over one million receptor-ligand complexes. A more complete solution to this problem involving a small scale optimised dynamics code and tiered scoring cut-offs to control the level of detail used in each calculation has been proposed, however its implementation lies beyond the scope of this thesis.

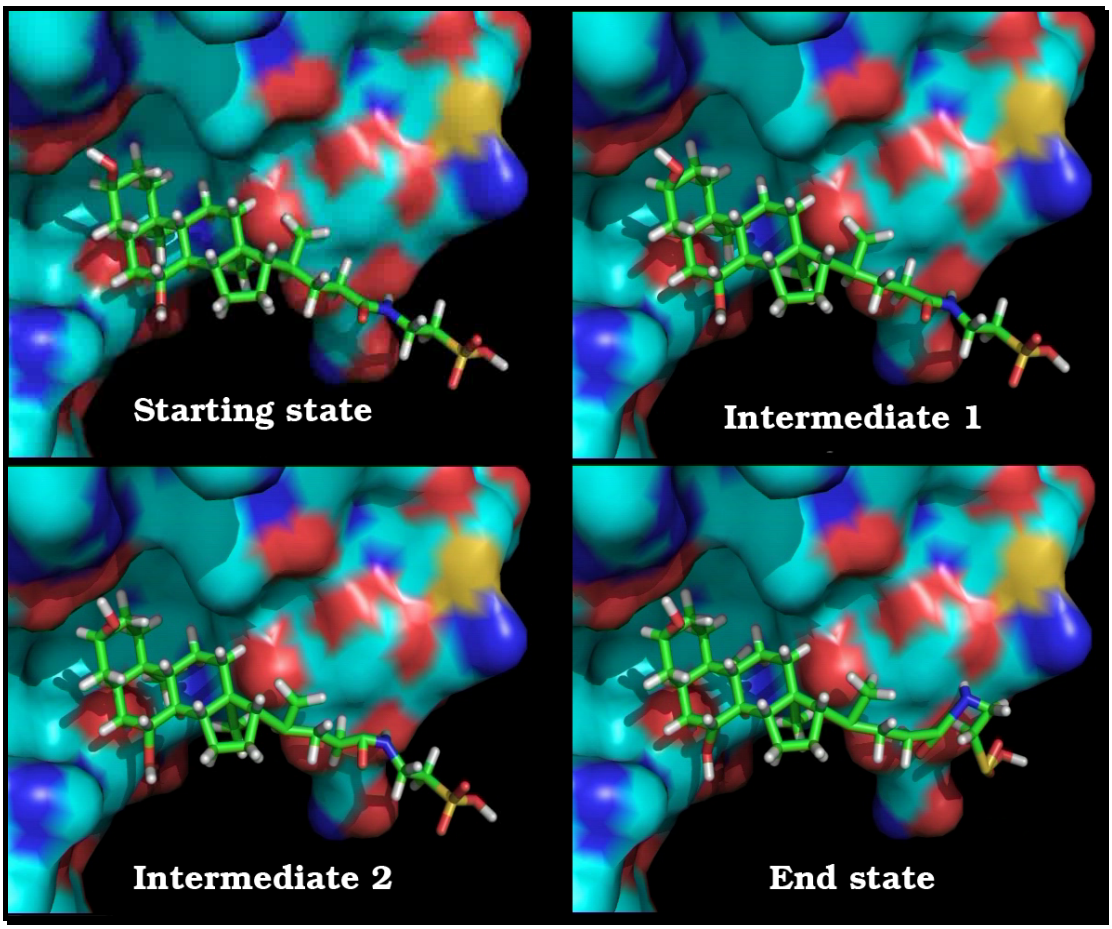


Figure 4.17 - Predicted conformational changes of a molecule as it binds to FKBP12 (PDBID 2DG3) . Predictions made using the simple flexible docking addition to LIDAEUS along with a simple steepest descent minimiser

5 The molecular similarity principle and ligand based virtual screening

The concept of molecular similarity is of great importance to biologists and chemists. The overall concept is that objectively similar molecules should behave in a consistent manner and have properties also in line with their similars. Whilst this seems obvious, the term ‘similar’ is extremely loose and can be defined in many different ways. This chapter builds on the molecular similarity principle and documents new tools for ligand based virtual screening developed during the course of this research. The tools have been successfully used to identify new ligands for a number of protein targets. These new ligands have been validated by colleagues in a biochemical laboratory.

Example similarity searches make use of protected alanine and protected tryptophan molecules. Here, the protective groups added to peptide bond atoms are methyls. The molecules N-methyl-alanine methyl ester and N-methyl-tryptophan methyl ester are referred to from here on as protected alanine and protected tryptophan.

5.1 Molecular similarity: a key technique in molecular informatics

Molecular similarity is a simple concept; one of many measures of similarity between two molecules is evaluated indicating the predicted degree of similarity between the two. The value of this comes from the quantified idea [155; 156; 157; 158; 159] that structurally similar compounds exhibit similar properties and may therefore bind to receptors by making the same or similar interactions [157; 160] . Molecular similarity is not only about identifying compounds which may show activity against a receptor (virtual screening).

Sheridan and Kearsley [161] identify three uses for molecular similarity:

> *"1. Clustering: grouping similar compounds together [158].*
>
> *2. Diversity: selecting a subset of disparate molecules from a larger set [162].*
>
> *3. Virtual screening."*

Another application is rational drug design, where medicinal chemistry approaches may be used to take a starting molecule and customise it by adding or removing groups within the molecule, changing it in the hope of increasing biological activity. The starting molecule will undoubtedly be 'similar' to the final tailored molecule.

The idea of molecular similarity has been exploited in nearly all chemical fields and has been used to great effect in the pharmaceutical industry to reduce the massive cost of drug development [163; 164; 165]. Molecular similarity enables a route to 'rescue' problematic compounds which may well be good inhibitors of a protein, but unsuitable for further development due to issues such as absorption, distribution, metabolism, excretion and toxicity (ADMET) [163; 166]. Numerous reasons for the current popularity of similarity methods have been put forward [161]. Many similarity methods offer the ability to carry out searches for compounds active against a receptor whilst little is known about the receptor itself, only molecules which bind, inhibiting it. Many similarity methods are computationally inexpensive, therefore allowing massive numbers of compounds to be screened. Sheridan postulates that more than one similarity method should be used for a reliable measure of 'overall' similarity and suggests ten diverse measures to be used to evaluate molecules [161].

5.1.1 The ideal similarity metric

The goal of molecular similarity methods is ultimately to establish a strong link between biological activity and the measure of similarity reported. Ideally molecular similarity techniques would be able to correlate experimental binding data with predicted values. This is a difficult task due to the large number of factors that may affect the way in which a ligand binds. One could imagine a simple similarity metric, where for example, a molecule must have two hydrogen bond donors spaced 15Å apart, and a donor between them. This could give a realistic indication of activity against a specific protein but would be unsuitable for almost every other case. In general, the effectiveness of the similarity method is dictated by the targeted ideal behaviour of the similarity metric and the ability to encode important information relating to activity. The ideal action of a similarity method is shown in Figure 5.1.

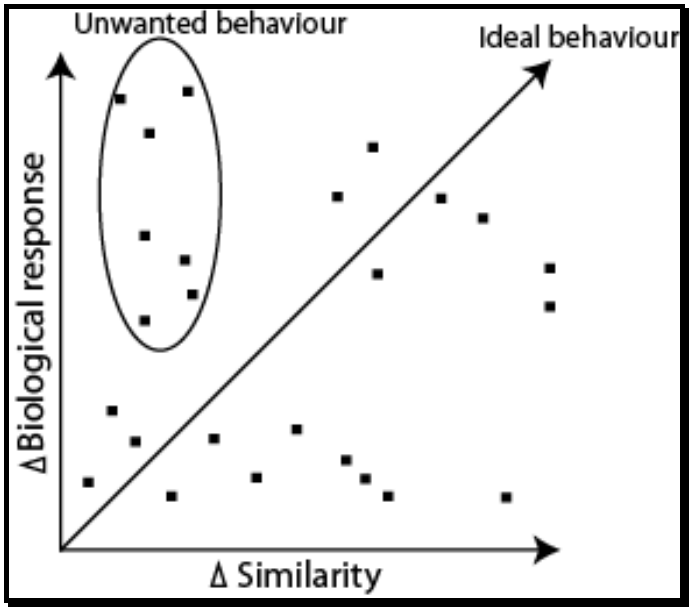


Figure 5.1 – "Illustration of neighbourhood behaviour which shows the relationship between the change in biological activity for pairs of a series of compounds plotted against the change in the descriptor for these pairs of compounds. The ideal behaviour would result in the lower triangle of the plot being occupied, with the most favourable outcome being that small changes in descriptor space are related to small changes in biological response (the typical medicinal chemistry paradigm). Small or large changes in biological response with large changes in descriptor space are also a common feature of such analysis. However, an unsuitable descriptor should not have the property that a large change in biological response results from small changes in the descriptor". [10]

Adapted from [164]

Numerous methods for calculating molecular similarity exist.

> *"… similarity has a context. Two vials of a yellow compound may be very similar in colour (absorption spectrum) but wildly different in biological activity. How far the context of a particular similarity argument can be taken (the 'neighbourhood effect') also depends on the discontinuities found in receptor-ligand interactions; clearly, the similarities studied are seldom linear and often have major discontinuities"* [164]

5.1.2 Quantitative structure-activity relationship

The simple concept of molecular similarity has been extended greatly and is the basis for Quantitative Structure-Activity Relationship (QSAR) approaches. As the name suggests, different techniques are used to find a link between the structural and activity related properties (usually biological activity, although could be used for solubility). The QSAR technique uses many methods available to correlate properties of compounds to biological activity. QSAR is used to create 'models', trained by knowledge of active compounds. A typical situation may be laboratory work finding biological actives, and measuring the response strength. With these actives, machine learning techniques are used to pick out from a vast number of molecular descriptors, features that are important in the desired biological activity. This computer model is then applied to a large number of compounds in an effort to predict further actives. When generating a model and training it with experimentally confirmed actives, it is important not to over-train the model. For example, the model may be so well trained that it only deems molecules in its training set to be active. If a large database of molecules is then put through it in this over-trained state, then it may see that only items present in the training set as fulfilling its criteria [167].

An obvious pitfall of similarity methods is that sometimes, molecules which may be subjectively deemed similar do not exhibit the same biological activity. The field of QSAR has given a name to this; the Structure Activity Relationship (SAR) paradox.

This states that ‘similar’ molecules should exhibit similar properties; but sometimes they do not. The ultimate aim of QSAR techniques is to find arelationship between some quantified and calculated property of molecules and their activity [168] which is not a simple task as it may employ properties from theoretical, derived or empirical. A number of good general reviews on QSAR exist in the literature [168; 169; 170; 171; 172; 173].

5.1.3 The many molecular similarity methods

Baringhaus [174] separates the way in which similarity methods represent molecules into three categories based on the dimensionality of information encoded, 1D, 2D or 3D.

1D

One dimensional information may include simple counts of atoms, the number of hydrogen bond donor atoms, log P and also linear representations of 2D and 3D properties such as surface area and Van der Waals/solvent accessible volume [160].

2D

Molecular similarity techniques encoding 2D information are the most widely accepted used class. The most common technique being binary fingerprints created from molecules and compared through some simple coefficient method. Binary fingerprints play a large role in bioinformatics. The technique commonly uses a set of fragments and analyses the query molecule to see which fragments exist within the

structure. If the fragment is found, then a ‘true’ value or ‘1’ is entered into the bit string representation of the molecule. The program openBabel can be used to generate binary fingerprints from molecules and calculate similarity by comparison using the Tanimoto coefficient. As with all aspects of similarity, there are many ways to perform each step; creating a binary fingerprint is no exception. Figure 5.2 shows an example of using the openBabel fingerprint type ‘fp3’ to generate the bit string for a pair of molecules which are then compared using the Tanimoto coefficient. Generating the bit string is a relatively simple task – with the fp3 fingerprint, the features that may be present in the molecule are assigned bit positions. A graph representation of the molecules is then searched and if it contains a certain feature, then the corresponding bit is set to true or ‘1’. Compound ‘a’ in Figure 5.2 has the following features which are encoded into its bit string at the appropriate positions: aldehyde or ketone, ether, dialkyl ether, thioether, aryl, heteroatom, hydrogen bond donor atom, and ring.

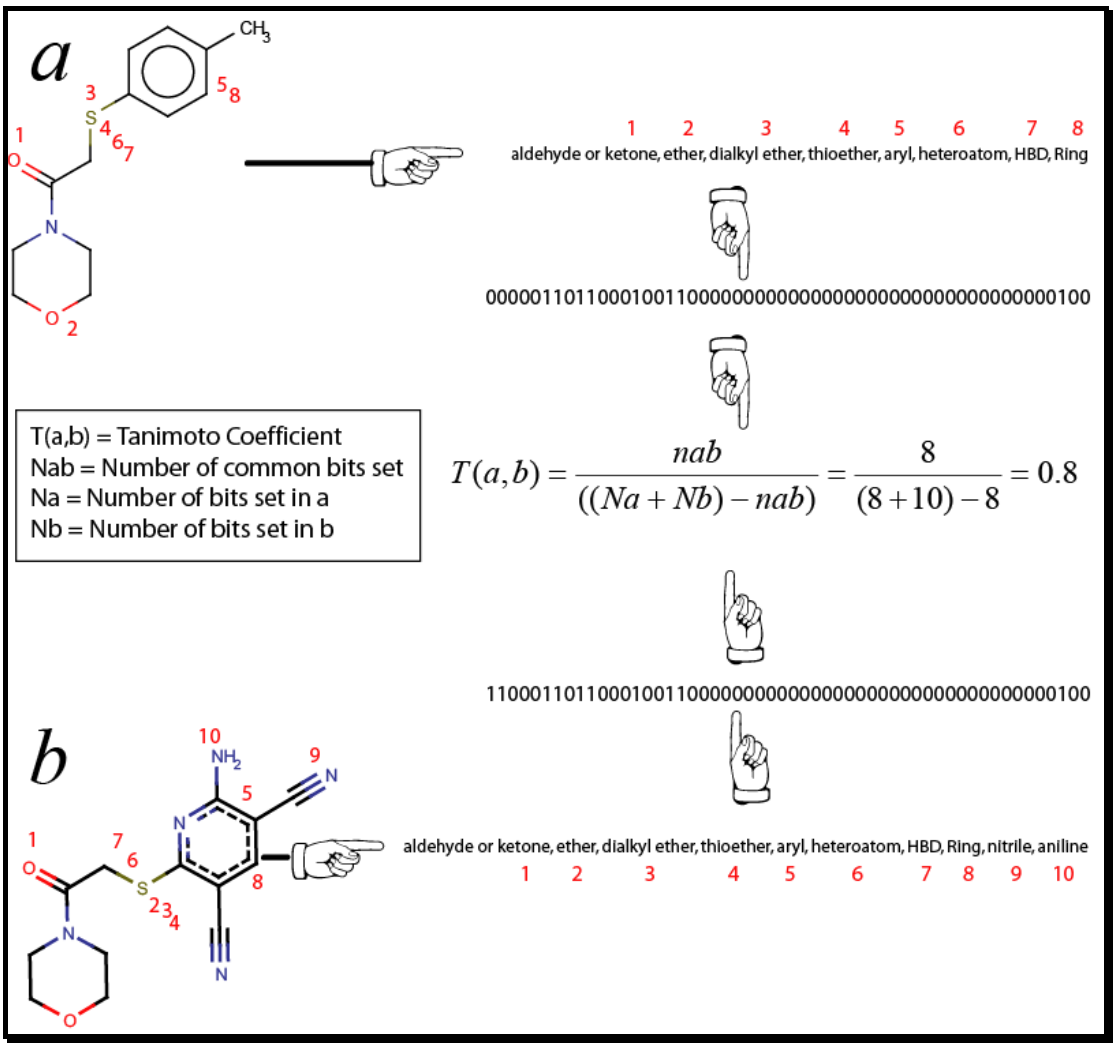


Figure 5.2 - Comparison of two molecules by their fp3 bit string and evaluated by the Tanimoto coefficient. Numbers identify molecular features. The presence of these features is encoded at appropriate positions within the 54 bit bit-string

The Tanimoto coefficient is the most well known scoring function used in molecular similarity. Attempts have been made at using other metrics, such as the cosine coefficient, which offers the ability to show both similarity and dissimilarity, returning a metric of the form:

$-1 \leq cosine.coefficient \leq 1$

This scoring function is perhaps not best suited to molecular similarity. If the function was used to evaluate two molecules and reported a score of -1, then this would suggest that the molecules were precisely opposite in nature. In reality, the bit string representation of one was the logical 'NOT' of the other. A score of 0 would imply that there was not a strong correlation between bits present in each molecules bit string. However, the coefficient could be normalised and its output made to lie in the range 0 to 1. This coefficient is scarcely used, bit string representations of molecules are compared almost exclusively using the Tanimoto coefficient. Many studies carried out show that Tanimoto outperforms all other methods. Holliday reviews coefficients for comparison of bit strings and identifies 22 methods, each used to compare a known dataset and groups the individual methods into clusters based on their discriminatory power [175]. Martin states that when using Tanimoto [176] similarity in combination with Unity fingerprints (another method of producing molecular bit string representations of molecules), a compound identified as greater than 85% similar to an active compound has itself an 80% chance of being active [160].

3D

Comparative field analysis (CoMFA) fields [9; 10; 11; 12; 177] are seen as the classic foundation of QSAR methods. The process involves obtaining a set of biologically active molecules against a target, using topomeric alignment techniques to overlay the molecules and then using a probe atom and sampling energies at intervals around the molecules. This map can then be used to infer the relevance of groups making interactions with the receptor. QSAR techniques may employ machine learning methods to automatically assign relevance to interactions or human expertise may be used. As mentioned briefly, topological alignment is necessary for this method. Other 3D techniques also require alignment. This can be troublesome when molecules are drastically different or do not share enough common features to make a distinct alignment.

In depth overviews of the many different existing similarity methods can be found [176; 178].

5.2 Ultra Fast Shape Recognition

Ultra Fast Shape Recognition (UFSR) [100; 179] is an algorithm that can be used to assess the similarity between two molecules. Applying the UFSR technique to a query molecule and a candidate molecule returns a number greater than 0 and less than or equal to 1. A similarity score of 1 being a perfect match and suggested similarity lessening as the number decreases and approaches 0. Comparing a query molecule against a candidate requires two steps: first shape descriptors are calculated for each molecule and second, the descriptors for each molecule are compared.

A descriptor set consists of a set of 12 values each calculated from geometric distributions created from the molecule. The 12 values can be grouped into 4 subsets with three values each. Calculating the 12 values takes the following form:

1. From the 3D coordinates of the molecule, calculate the centre of the volume occupied by the molecule.
2. The centre of the volume is defined as point 1 (P1).
3. A list of Euclidean distances of all atoms to P1 is generated.
4. From this list, 3 values are generated; mean, variance and skew. These three values for all atoms relative to P1 make up the first three of the twelve descriptors. This can also be called a set as they are all describing the geometric distribution of atoms around P1.
5. The closest atom to the geometric centre is called point 2 (P2). Similarly, the Euclidian distance of all atoms to P2 is calculated and from this list of

distances, the mean, variance and skew calculated. These make up descriptor values 4, 5 and 6.

6. The furthest atom from P2 is point 3 (P3). Again, the mean, variance and skew of the distribution made up from the Euclidian distance of all atoms from P3 go to make up values 7, 8 and 9 of the descriptors
7. Finally, point 4 (P4) is the furthest atom from P3. To make up the final three values of the twelve descriptors, the Euclidian distance of all atoms from this point is calculated and then the mean, variance and skew calculated from this distribution.

Figure 5.3 shows the UFSR process in simplified 2D applied to an arginine molecule.

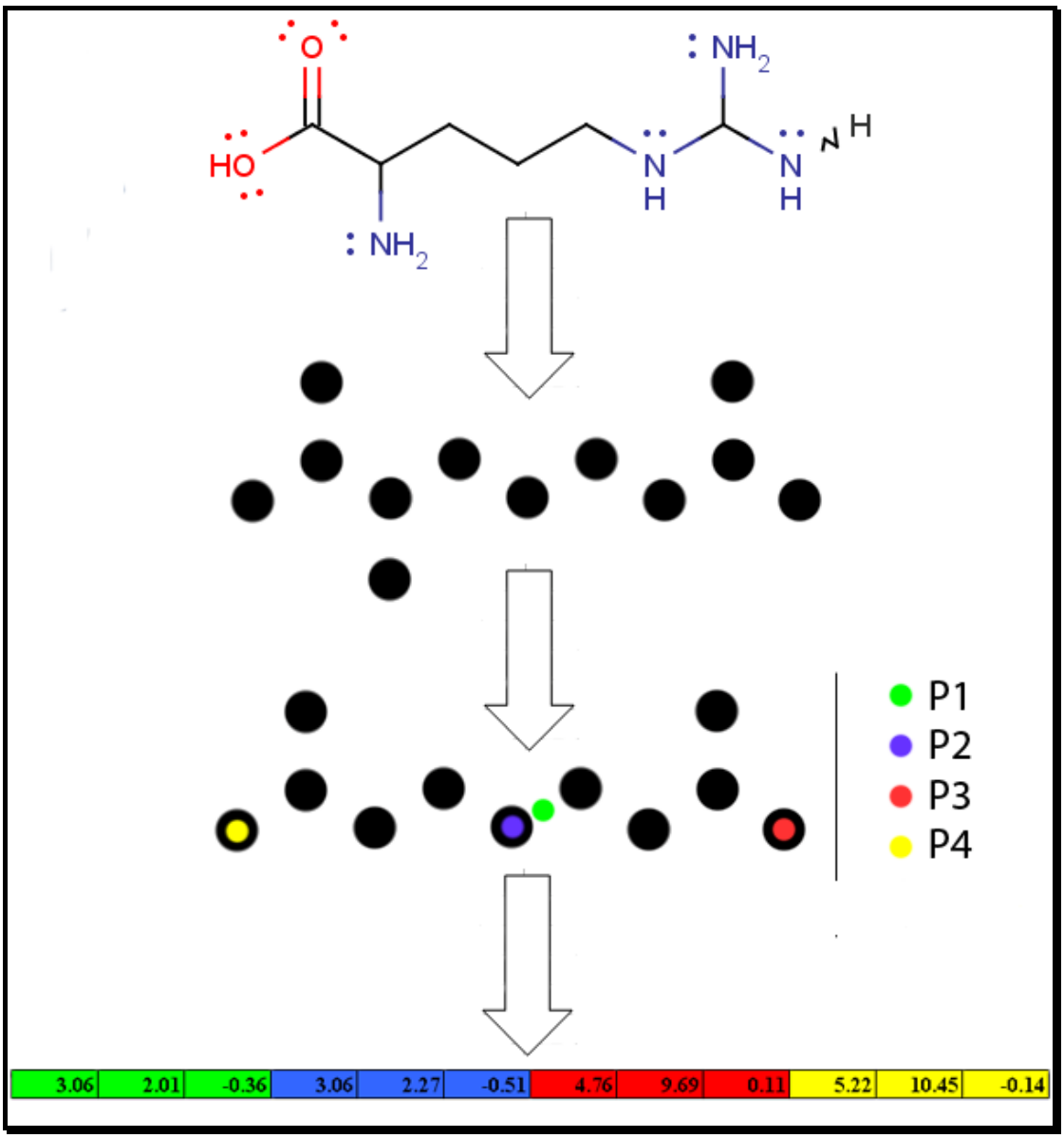


Figure 5.3 - Generation of UFSR descriptors. The molecules atoms are all considered points within a distribution. Starting with the geometric centre (P1), all atom distances are calculated from this point. From these values, the mean, variance and skew of the created distribution are calculated and make up the first 3 descriptors. The same process is repeated three more times, using the closest atom to P1 (which is P2). P3 is defined as the furthest atom from P2. Finally P4 is the furthest atom from P3

The next step is to compare the molecules. UFSR uses the previously generated descriptors for each molecule and applies a scoring function to them which is shown in Figure 5.4.

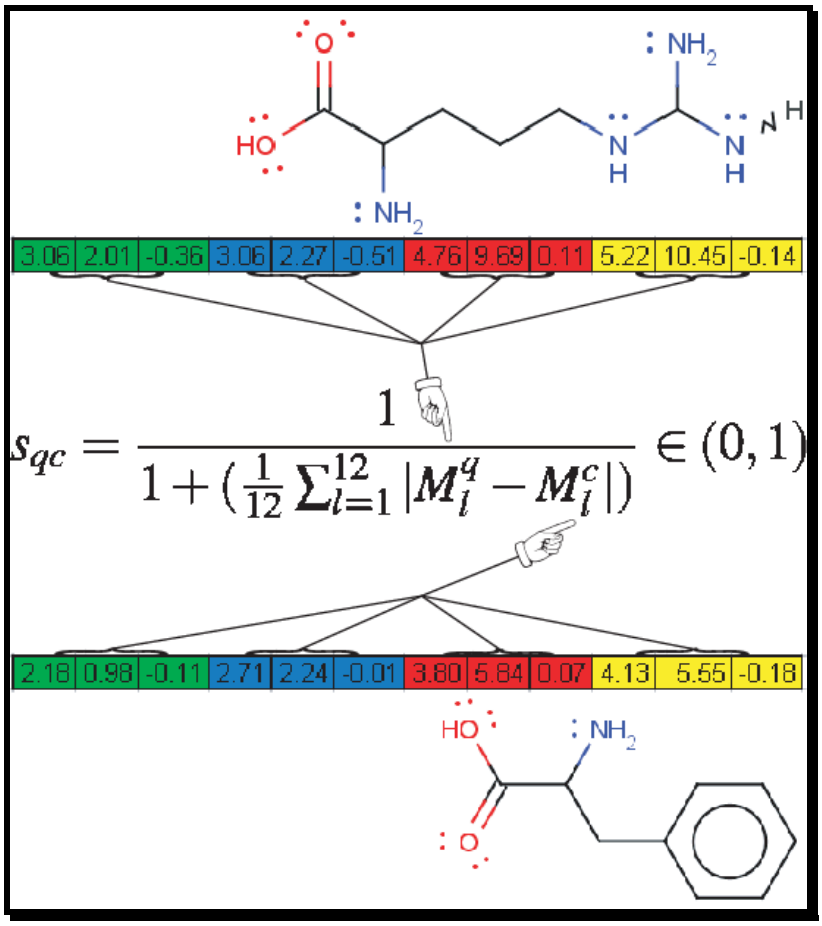


Figure 5.4 - Comparison of two molecules by their UFSR descriptors (UFSR Scoring function). UFSR descriptor values have been calculated for the molecules shown and their values are input to the scoring function shown

S_{qc} is the similarity score between the two molecules q and c (query and candidate), M being a vector representing the 12 geometric distribution descriptors.

Carrying out the calculation above yields the result $S_{qc} = 0.506$. It is not possible to assign any significance to the value of ~0.5 by itself, as a score of 0.8 or greater signifies a good match may be appropriate for one query molecule but not another. Assigning meaning to the discriminatory power of this value requires comparison to other candidate molecules. Therefore, UFSR is typically used to screen a large

database of potential candidate molecules against a query resulting in a ranked list of arbitrary length such as the top 500 similars.

An implementation of UFSR has been developed from the papers documenting the algorithm. The implementation has been developed in the C++ programming language and deployed on local Linux systems.

5.2.1 UFSR Pseudo code

```
Read in query molecule
Define geometric centre of the molecule - P1
Calculate list of Euclidean distances of atoms from P1
From Euclidean distance list calculate mean, variance and skew of distribution
these are query descriptors 1 to 3
Define closest atom to the molecular centroid - P2
Calculate list of Euclidean distances of atoms from P2
From Euclidean distance list calculate mean, variance and skew of distribution
these are query descriptors 4 to 6
Define furthest atom from P2 - P3
Calculate list of Euclidean distances of atoms from P3
From Euclidean distance list calculate mean, variance and skew of distribution
these are query descriptors 7 to 9
Define furthest atom from P3 - P4
Calculate list of Euclidean distances of atoms from P4
From Euclidean distance list calculate mean, variance and skew of distribution
these are query descriptors 10 to 12
While candidate molecules can still be read in:
       Read in candidate molecule
       Define geometric centre of the molecule - P1
       Calculate list of Euclidean distances of atoms from P1
       From Euclidean distance list calculate mean, variance and skew of
distribution these are candidate descriptors 1 to 3
       Define closest atom to the molecular centroid - P2
       Calculate list of Euclidean distances of atoms from P2
       From Euclidean distance list calculate mean, variance and skew of
distribution these are candidate descriptors 4 to 6
       Define furthest atom from P2 - P3
       Calculate list of Euclidean distances of atoms from P3
       From Euclidean distance list calculate mean, variance and skew of
distribution these are candidate descriptors 7 to 9
       Define furthest atom from P3 - P4
       Calculate list of Euclidean distances of atoms from P4
       From Euclidean distance list calculate mean, variance and skew of
distribution these are candidate descriptors 10 to 12

       For each value in descriptor sets, find difference between query and
candidate and sum
       Divide sum by 12 to achieve an average difference for each descriptor value
       Add one to average
       Take the reciprocal of the average to achieve a UFSR score
       Add SIMILARITY key to query molecule documenting the score achieved
       If score is good enough to be kept in the top X, then add the molecule in
the correct position to the list to be kept
       If the list to be kept is too big, remove the lowest scoring molecule from
the list.
End while
Loop over list to be kept outputting suitable candidate molecules
```

5.2.2 Common Chemicals and their UFSR predicted similars

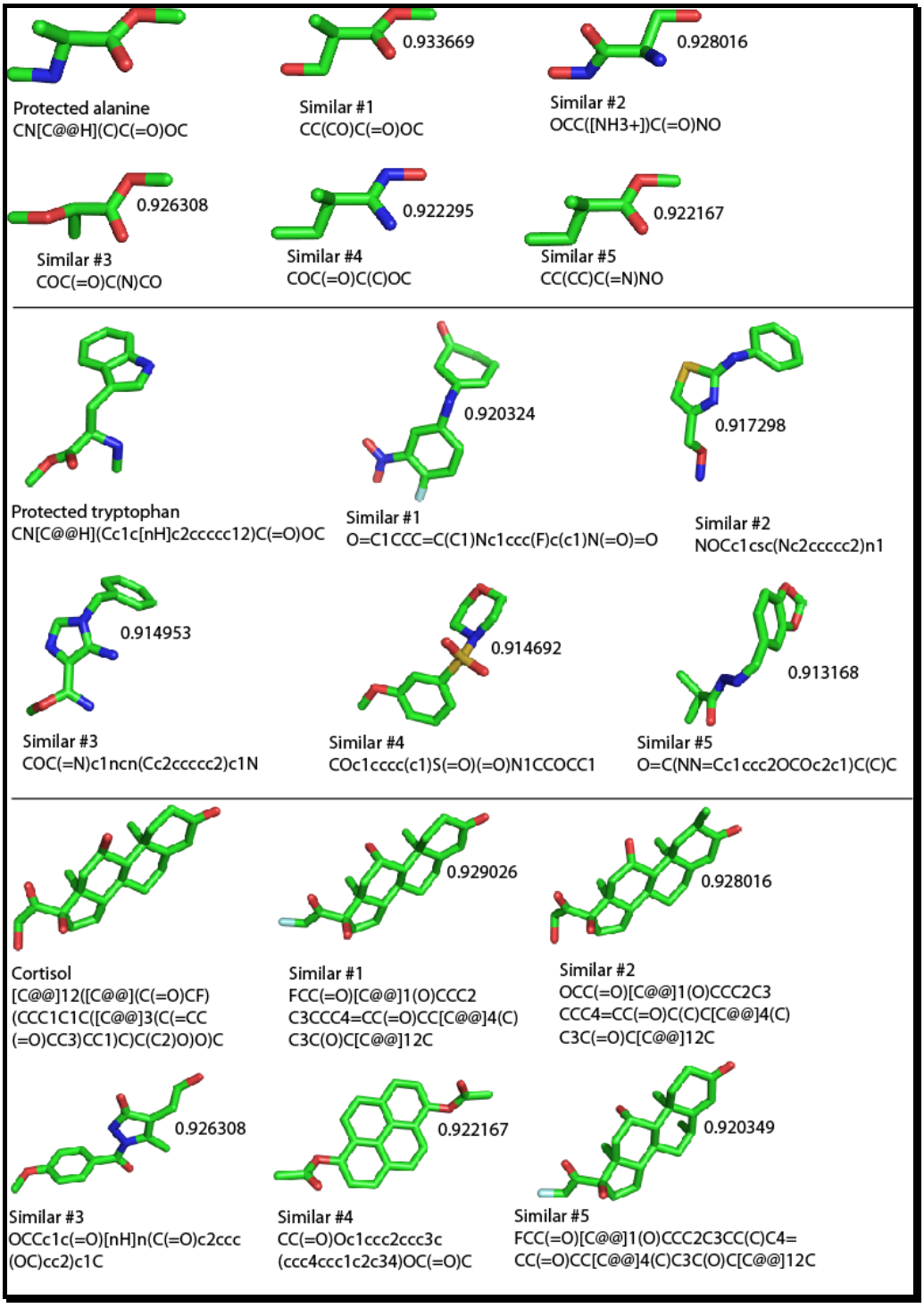


Figure 5.5 – Protected alanine, protected tryptophan and cortisol UFSR predicted similars with scores and SMILES representations. Molecules scored and similars searched from EDULISS dataset of 3.67 million molecules

The common chemicals chosen are protected alanine, protected tryptophan and cortisol. The amino acids alanine and tryptophan have had protective groups added to their reactive positions giving them the ‘protected’ prefix. This will help matching as the EDULISS database contains purchasable compounds, the reactive groups on molecule contained within it will likely be protected. Figure 5.5 shows these common chemicals and their UFSR predicted similars.

A subjective visual inspection remembering the limitations in the way in which UFSR works confirms that good similars have been found. Inspection shows that the overall shape of the top candidate molecules has remained consistent with the queries in all but possibly similar 3 of cortisol. The query molecule protected alanine shows UFSR doing exactly what it has been developed to do in pulling out its similars; matching the overall shape of the query. In evaluating its effectiveness and confirming the implementation was successful, the atom types must be ignored. UFSR considers only one atom type, all atoms being identical. Nitrogen is no different to carbon or oxygen. Considering this limitation, UFSR has been successful in pulling out similarly shaped molecules.

5.2.3 UFSR Profiling

This section benchmarks the performance of the UFSR code on standard hardware. Two datasets are used.

5.2.3.1 Dataset profiling

From the EDULISS database, the catalogues from the two suppliers Specs and Sigma were chosen and the compounds present in each extracted into two files following the SDF format. These two catalogues are used as experimental datasets for benchmarking purposes. A small custom written C++ program has been developed and used to profile the datasets. Reading in the dataset, information is collected on the number of atoms present in each molecule and then statistics calculated. Counts of the number of molecules containing certain numbers of atoms is also output in a comma separated list allowing the easy import of data into spreadsheet and other analysis programs.

The Specs dataset contains a total of 226,364 molecules and the following properties:

- Average atoms per molecule = 45.329
- Smallest molecule = 7 atoms
- Largest molecule = 131 atoms
- Standard deviation present in number of atoms per molecule distribution = 11.4
- Skew present in number of atoms per molecule distribution = 0.45

Figure 5.6 shows a profile of the Specs dataset, specifically how many molecules contain a certain atom count.

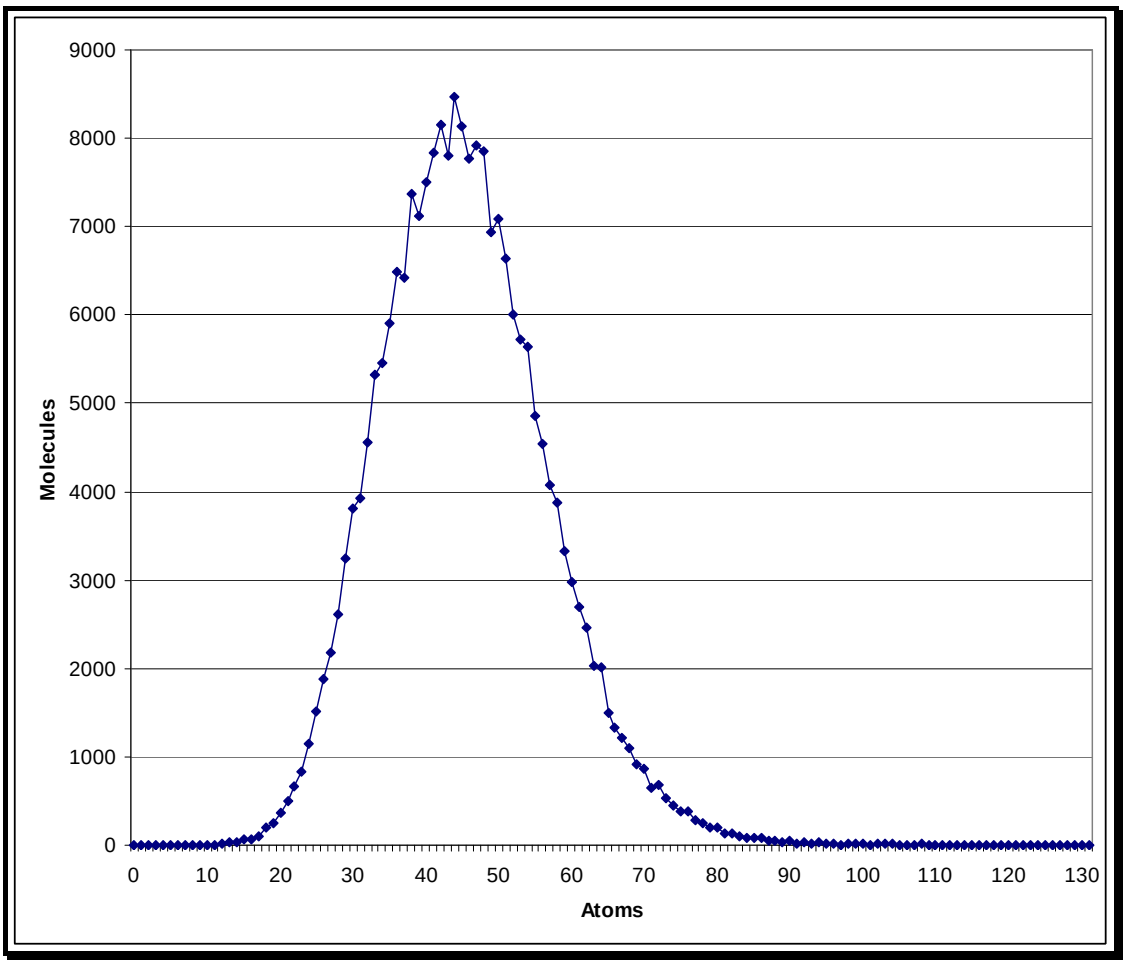


Figure 5.6 – Profile of the Specs dataset, number of molecules present with specific numbers of atoms

The Sigma dataset contains a total of 199,492 molecules and the following properties:

- Average atoms per molecule = 39.2
- Smallest molecule = 2 atoms
- Largest molecule = 361 atoms

- Standard deviation present in number of atoms per molecule distribution = 17.53

- Skew present in number of atoms per molecule distribution = 1.01

Figure 5.7 shows the profile of the Sigma dataset and counts of molecules containing certain numbers of atoms (displaying counts of 1-130 only).

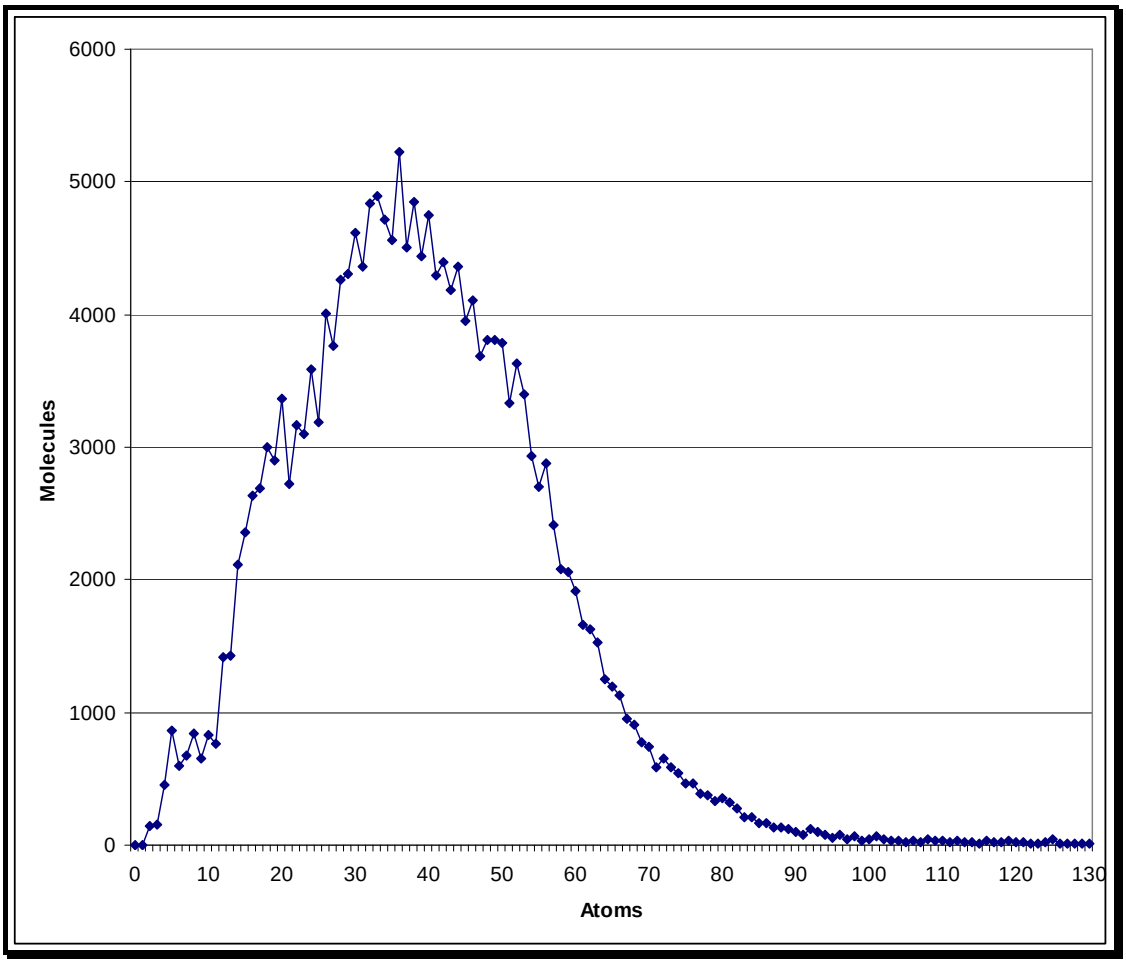


Figure 5.7 - Profile of the Sigma dataset, number of molecules present with specific numbers of atoms. Displaying only counts of 1 atom to 130 atoms.

5.2.3.2 UFSR performance

The protected tryptophan molecule was used as the query for UFSR benchmarking and run against the Sigma and Specs catalogues as profiled above. Runs were carried out on a dual core Intel Xeon 3050 machine with a clock speed of 2.13 GHz, 4GB ram and running Linux kernel 2.6.18.2-34. Although the machine is dual core, UFSR makes use of only one execution thread and therefore uses one core only.

Execution time with the Specs catalogue providing the candidate molecules was 6 minutes 9 seconds. This equates to roughly 613 candidate-query comparisons per second. 1 million comparisons are therefore predicted to take ~27 minutes.

Execution time with the Sigma catalogue providing the candidate molecules was 4 minutes 37 seconds. This equates to roughly 720 candidate-query comparisons per second. 1 million comparisons are therefore predicted to take ~23 minutes.

The difference in execution time for the two datasets can be put down to average molecule size contained within each. Although the Specs set contains fewer molecules, the average number of atoms has increased from 39.23 to 45.32; an increase of ~15% whilst the difference in number of molecules between Sigma and Specs is 13%. To look at it another way, the Sigma dataset contains ~8.9 million atoms whilst Sigma contains 7.8 million atoms. As UFSR is concerned solely with

atom positions, the Sigma dataset takes less time to process. This explanation of speedup is a simplification as the code for reading in molecules and typing atoms has to be run more times. However, the code for calculating UFSR distributions is of the order 4N on atoms meaning that the average number of atoms contained within molecules within the dataset will dominate execution time.

Figure 5.8 shows a UFSR run using candidate molecules from the entire EDULISS database (~1.67 million compounds) and shows the score distribution achieved with three query molecules (protected alanine, protected tryptophan and cortisol). **Error! Reference source not found.** gives the raw counts

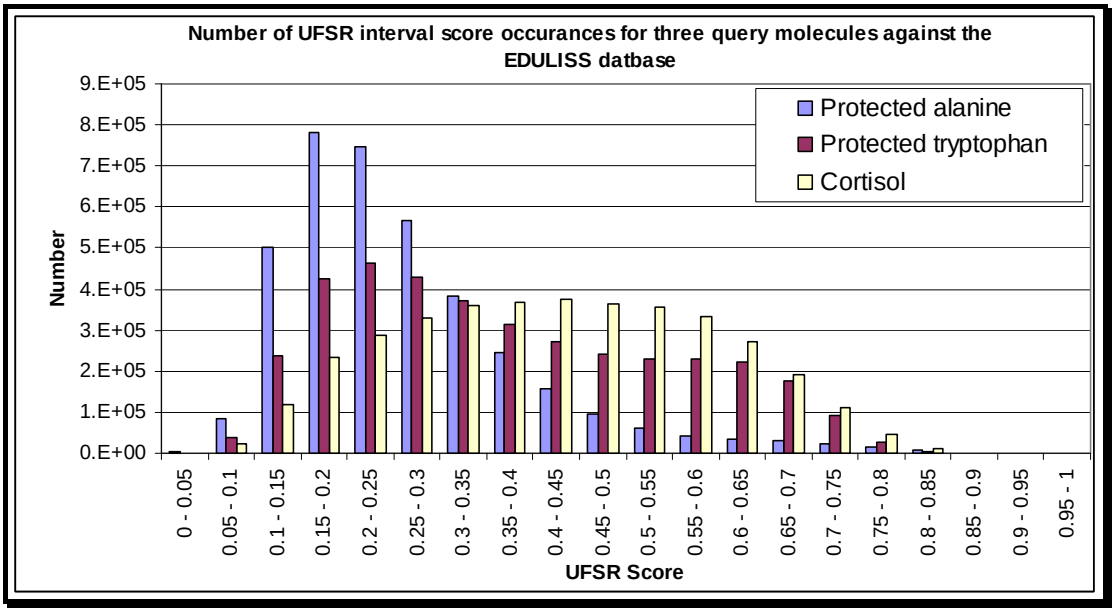


Figure 5.8 – Number of UFSR interval score occurrences for three query molecules against the EDULISS database. Score distribution and the discriminatory abilities of UFSR can be seen

UFSR Score	Protected Alanine	Protected Tryptophan	Cortisol
0-0.05	2,174	1,599	1,279
0.05-0.1	82,916	38,682	21,079
0.1-0.15	501,442	235,951	119,166
0.15-0.2	780,768	424,764	234,212
0.2-0.25	748,169	463,409	286,945
0.25-0.3	567,212	430,114	328,357
0.3-0.35	382,253	371,529	358,153
0.35-0.4	244,264	314,496	367,848
0.4-0.45	155,419	271,596	373,737
0.45-0.5	94,950	242,762	362,977
0.5-0.55	61,519	230,820	355,856
0.55-0.6	43,130	231,318	332,297
0.6-0.65	35,963	221,310	272,148
0.65-0.7	30,631	175,712	192,487
0.7-0.75	23,173	92,127	112,276
0.75-0.8	16,017	27,236	46,974
0.8-0.85	6,793	4,519	11,099
0.85-0.9	1,435	431	1,456
0.9-0.95	138	14	43
0.95-1	23	0	0

Table 5.1 - Number of UFSR interval score occurrences for three query molecules against the EDULISS database. Score distribution and the discriminatory abilities of UFSR can be seen

5.3 Ultra Fast Shape Recognition with Atom Types (UFSRAT)

Ultra Fast Shape Recognition with Atom Types (UFSRAT) has been developed using UFSR as the base concept. The obvious major shortcoming of UFSR is that no distinction is made between atoms of different type, each being treated identically (except hydrogen which is ignored). Treating all atoms as points encodes only partial hydrophobic information. UFSRAT has been developed to overcome this limitation, encoding further features of the molecule important in biochemistry.

The current implementation of UFSRAT makes use of not only the standard UFSR distribution made up of all atoms, but also combines three others. The distributions used to generate UFSRAT descriptors are:

1. All atoms
2. Hydrophobic
3. Hydrogen bond acceptor
4. Hydrogen bond donor

Determining which atoms within the molecules should be considered for each of the distributions requires atom type information to be calculated. This is achieved using the same implementation as present in the high throughput virtual screening code LIDAEUS [107; 127; 180]. Once atom type information has been established, the distributions can be calculated, essentially using atom types as a flexible user

definable mask on the condition of whether an atom is considered in each distribution. 12 descriptors in each distribution results in UFSRAT using 48 descriptors to describe each molecule encoding biologically relevant information. Figure 5.9 shows the generation of the UFSRAT distributions.

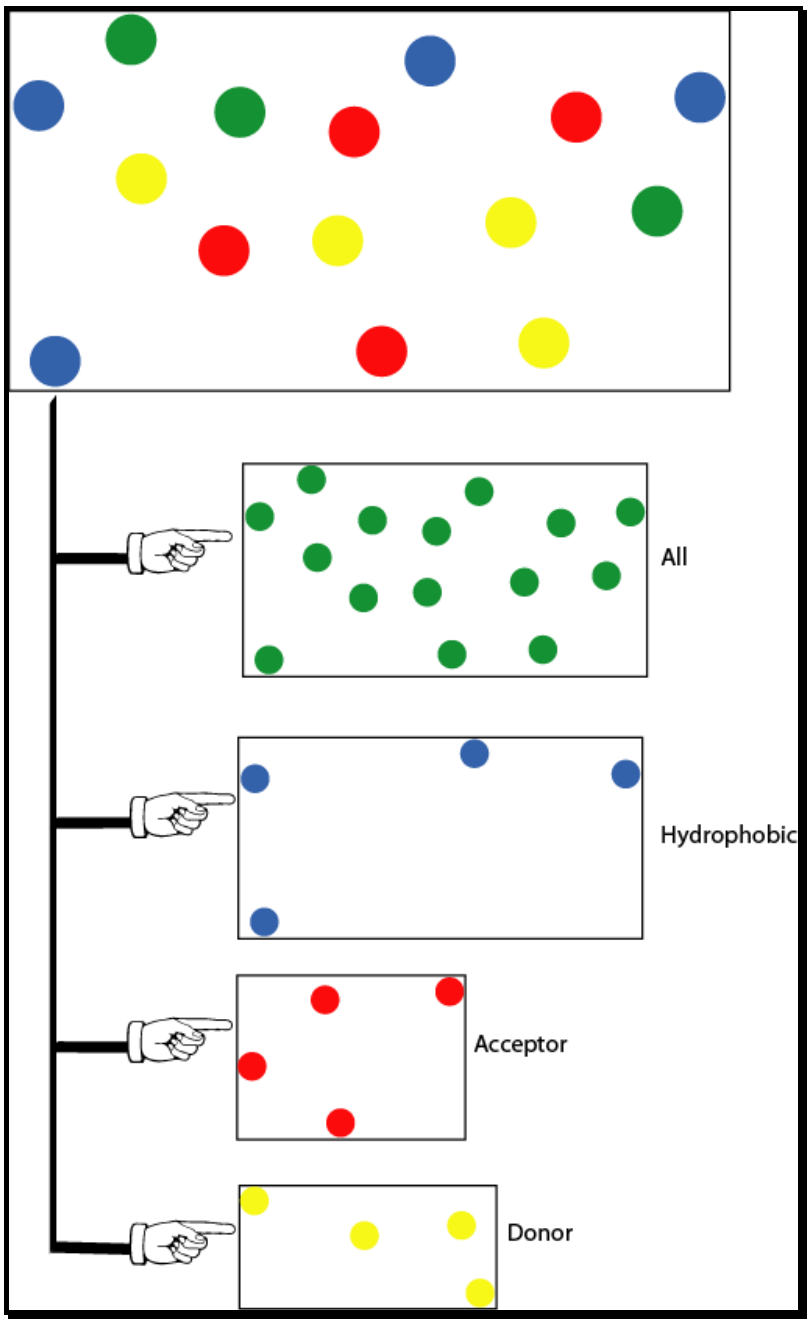


Figure 5.9 – The four UFSRAT distributions from typed atoms. A molecule is essentially broken down into four distributions, consisting of all atom, hydrophobic only, hydrogen bond acceptor only and hydrogen bond donor only distributions

The mode of operation of UFSRAT is similar to UFSR in that the query molecules geometric distribution descriptors are calculated along with the candidate's, and then

a scoring function is used to derive a single numerical metric pertaining to the predicted similarity of the molecule (0 < score <=1). As candidate molecules are tested against the query, a sorted list of the top matches is kept and returned as the result upon completion. The scoring function is an extension of the method used by UFSR, calculating a weighted difference for each distribution and is shown in Figure 5.10.

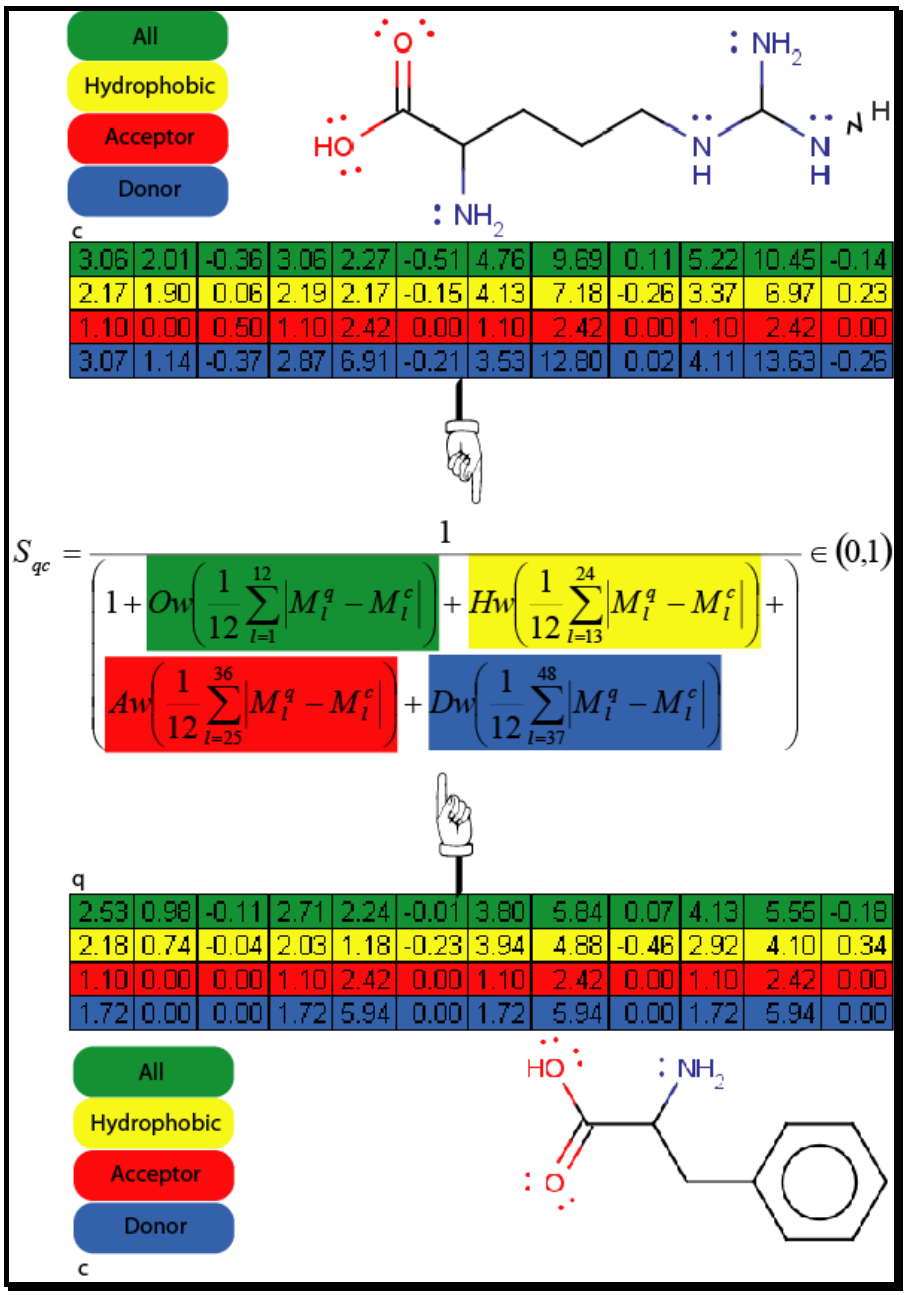


Figure 5.10 - Comparison of two molecules by their UFSRAT descriptors (UFSRAT Scoring function). UFSRAT descriptor values have been calculated for the molecules shown and their values are input to the scoring function shown

S_{qc} is the similarity score between the two molecules q and c (query and candidate), M being a vector representing the 48 geometric distribution descriptors, twelve for each of the four distributions. An optional weighting can be added to each of the

four distributions by the use of the Ow, Hw, Aw and Dw variables. Deviation from the default values of 1 for each of these may cause the assertion $\epsilon(0,1)$ to become false.

5.3.1 UFSRAT pseudo code

```
Read in query molecule

Type atoms using LIDAEUS typing system

Create four distributions containing points for all atoms, hydrophobic atoms,
hydrogen bond acceptor atoms and hydrogen bond donor atoms.

Calculate UFSR descriptors for each of these four distributions, giving (4 x 12) 48
descriptors which make up a set of UFSRAT descriptors.

While candidate molecules can still be read in:

        Read in candidate molecule.

        Type atoms using LIDAEUS typing system

        Create four distributions containing points for all atoms, hydrophobic
        atoms, hydrogen bond acceptor atoms and hydrogen bond donor atoms.

        Calculate UFSR descriptors for each of these four distributions, giving (4
        x 12) 48 descriptors which make up a set of UFSRAT descriptors.

        For each value in 4 descriptor sets of 12 UFSR distributions, find
        difference between query and candidate and sum, divide by 12 and apply
        appropriate weighting the cumulative contribution from the 4 distributions
        is the score.

        Add one to the score

        Take the reciprocal of the score to achieve a UFSR score

        Add SIMILARITY key to query molecule documenting the score achieved

        If score is good enough to be kept in the top X, then add the molecule in
        the correct position to the list to be kept

        If the list to be kept is too big, remove the lowest scoring molecule from
the list.

End while

Loop over list to be kept outputting suitable candidate molecules
```

5.3.2 Common chemicals and their UFSRAT predicted similars

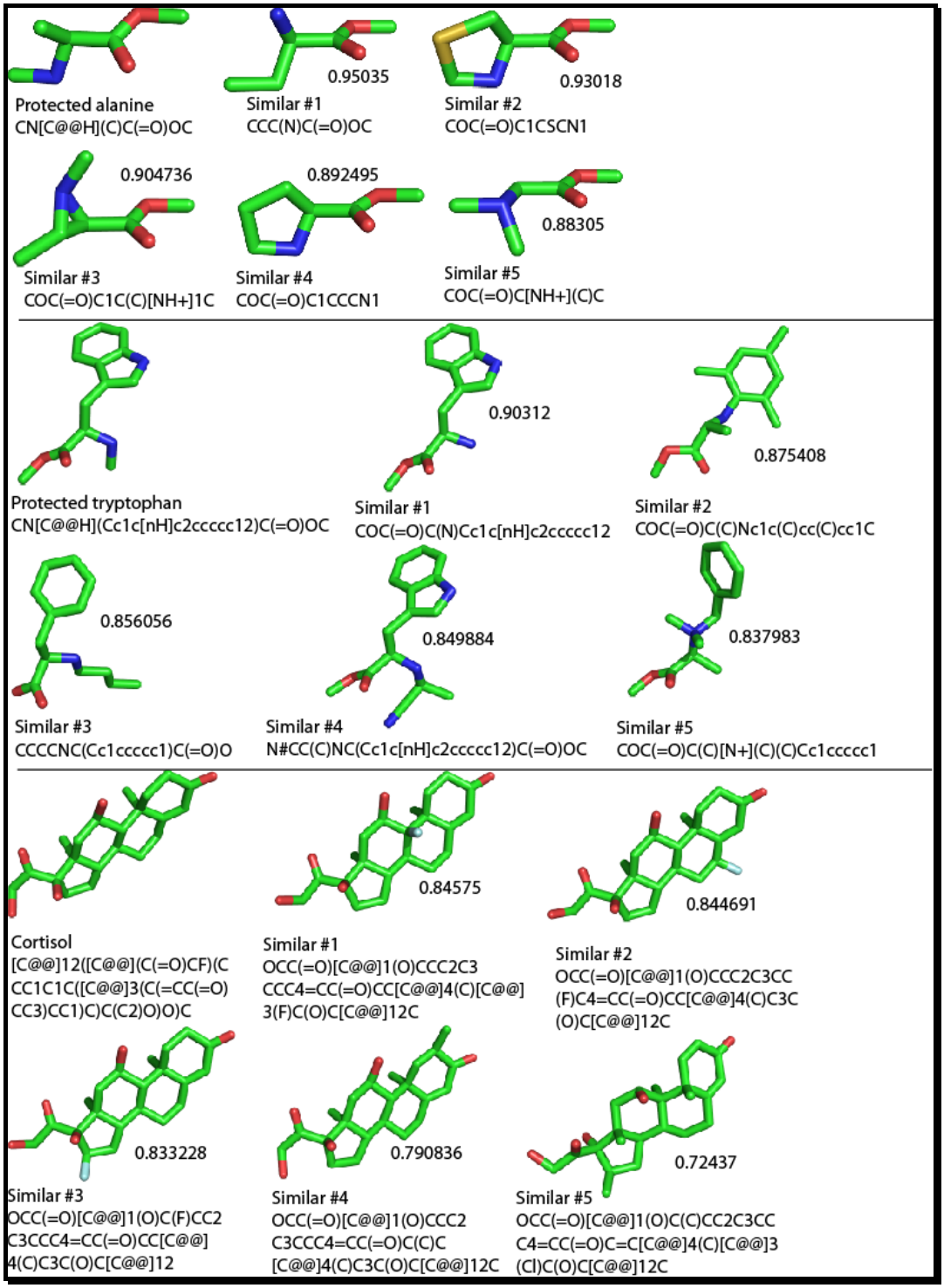


Figure 5.11 – Protected alanine, protected tryptophan and cortisol UFSRAT predicted similars with scores and SMILES representations. Molecules scored and similars searched from EDULISS dataset of 3.67 million molecules

Subjective visual inspection of Figure 5.11 shows that UFSRAT has predicted similars with a high degree of shape similarity and atom type/position similarity. Due to the way UFSRAT is implemented, and in contrast to UFSR, the overall shape of the molecule contributes less to the similarity score – weighting is given for atoms of certain type being in common positions. This is most evident with the protected tryptophan similars, which show that overall shape has been given a lesser importance than the position of key atoms. This is important to ensure that similars are able to make the same interactions within biological systems as the query used to predict them.

5.3.3 UFSRAT Profiling

UFSRAT profiling has been carried out in the same way as in section 5.2.3 – UFSR profling, using the same supplier catalogues from Specs and Sigma and the same hardware. This section provides a benchmark and gives an idea as to the complexity and time taken for the program to find similar molecules.

Execution time with the Specs catalogue providing the candidate molecules was 14 minutes 25 seconds. This equates to roughly 262 candidate-query comparisons per second. 1 million comparisons are therefore predicted to take ~64 minutes.

Execution time with the Sigma catalogue providing the candidate molecules was 4 minutes 37 seconds. This equates to roughly 305 candidate-query comparisons per second. 1 million comparisons are therefore predicted to take ~55 minutes.

As mentioned in section 5.2.3, the UFSR algorithm is of order 4N. Whilst the order of UFSRAT is complicated by the different distributions used it can be approximated to order 4N on atoms also (depending somewhat on atom types encountered). UFSRAT obviously adds complexity to the calculation and this is reflected in the runtimes observed. An interesting observation is that in the case of UFSR, the Sigma catalogue took 85.1% of the Specs runtime to complete all comparisons. This effect

is present here in the UFSRAT profiling runs, Sigma taking 85.9% of the Specs runtime.

Figure 5.12 shows a UFSRAT run using candidate molecules from the entire EDULISS database (~1.67 million compounds) and shows the score distribution achieved with three query molecules (protected alanine, protected tryptophan and cortisol). Counts data of the similar scores is available in

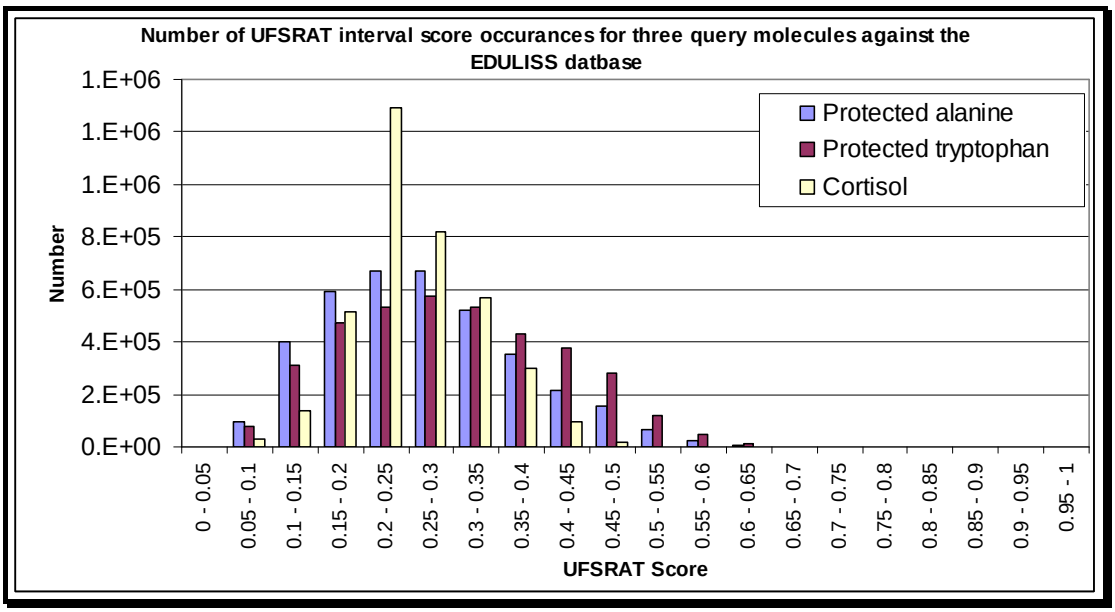


Figure 5.12 - Number of UFSRAT interval score occurrences for three query molecules against the EDULISS database. Score distribution and the discriminatory abilities of UFSRAT can be seen

UFSRAT Score	Protected Alanine	Protected Tryptophan	Cortisol
0-0.05	2,750	2,397	1,565
0.05-0.1	98,561	77,534	32,065
0.1-0.15	398,606	310,629	139,128
0.15-0.2	590,169	474,771	512,799
0.2-0.25	670,413	531,369	1,290,368
0.25-0.3	671,380	571,502	821,474
0.3-0.35	522,587	534,459	566,302
0.35-0.4	354,109	430,445	298,666
0.4-0.45	215,563	379,676	95,765
0.45-0.5	156,993	283,655	17,898
0.5-0.55	63,534	119,110	2,193
0.55-0.6	23,145	47,267	139
0.6-0.65	8,181	13,403	14
0.65-0.7	1,846	1,852	5
0.7-0.75	430	307	8
0.75-0.8	111	10	0
0.8-0.85	9	3	0
0.85-0.9	2	0	0
0.9-0.95	0	0	0
0.95-1	0	0	0

Table 5.2 – Number of UFSRAT interval score occurrences for three query molecules against the EDULISS database. Score distribution and the discriminatory abilities of UFSRAT can be seen

5.3.4 Hypothetical UFSRAT ligand descriptors from protein structure

UFSRAT and other similarity methods offer the attractive prospect of finding ligands for a receptor when knowledge of other ligands of the binding site are also known. This is valuable when no 3D data describing the protein is available. Similarity methods are not normally concerned with taking structural information on the target receptor and using this to find potential ligands in the way that classic virtual screening is used. An extension to UFSRAT has been implemented and given a 3D representation of a protein in the PDB file format aims to generate a set of descriptors that would describe a suitable ligand to fill the pocket and make key hydrogen bonding interactions. Firstly, atoms would be placed in energetically favourable positions and then descriptors generated. It would then be possible to undertake a standard UFSRAT run comparing the potential ligands with this theoretically ideal set of descriptors. Figure 5.13 illustrates the aim of this approach, showing a demonstration 'ideal' set of points or atom positions. In this example, the points were chosen for illustration purposes and taken from the molecule FK506 bound to the FKBP12 protein (See section 5.3.6 Prediction and validation of FKBP12 ligands using UFSRAT). It was the intention that the developed program would be able to define a set of points similar to this with only knowledge of the protein structure.

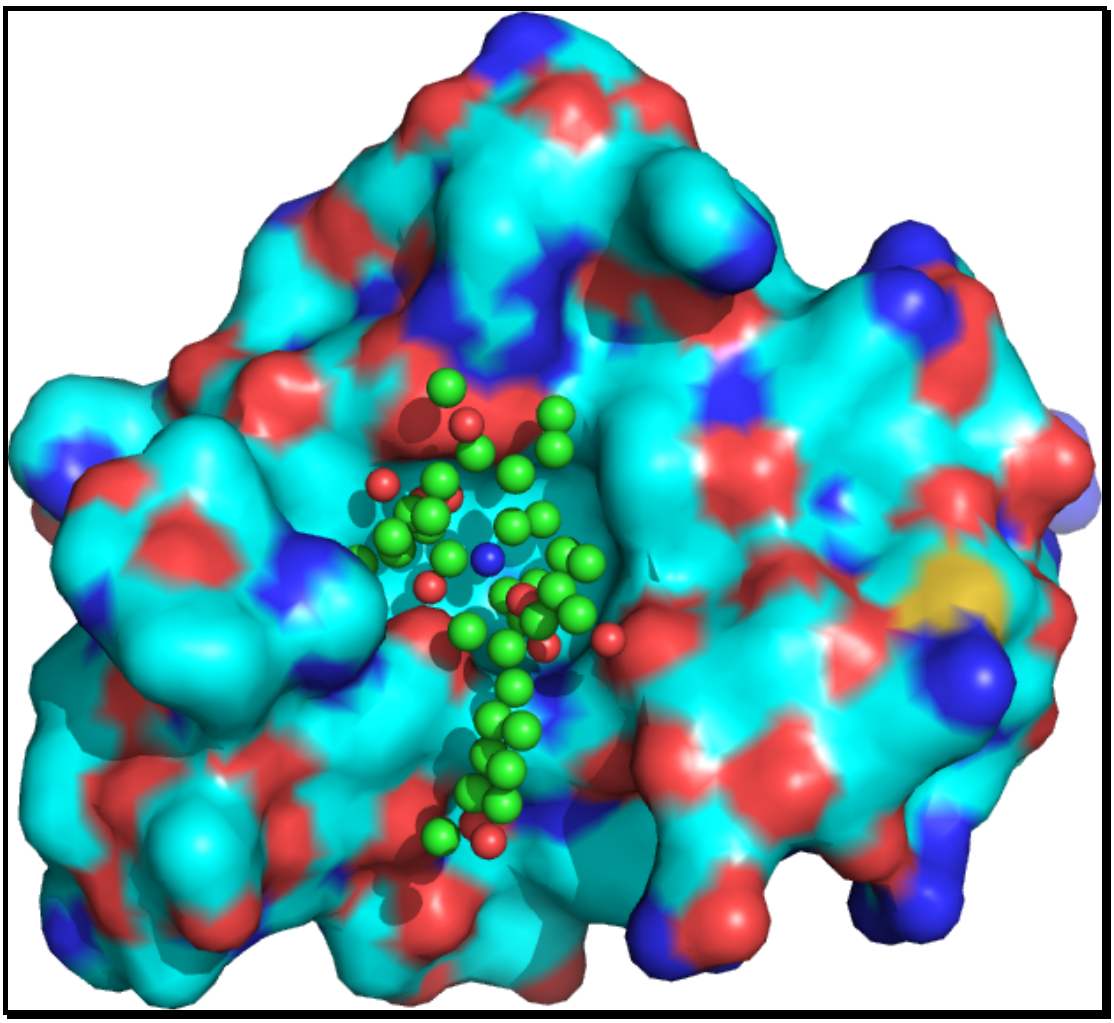


Figure 5.13 - The aim of generating hypothetical UFSRAT descriptors. Mock-up of ideal simulation results, FKBP12 (PDB ID 1FKJ) is shown with UFSRAT points made from FK506 ligand

The implementation uses the pre-processing steps present in the early stages of a LIDAEUS run. With a PDB file of a protein, the standard LIDAEUS energy maps are generated describing the energies felt by different types of probe atoms at regularly spaced intervals within and surrounding the volume occupied by the protein. With a dummy molecule inserted into the binding site of interest, a bounding box is generated. The volume within this box will define the area where

the predicted ideal distribution will be generated. For the remainder of this discussion, the volume will refer to the space defined by this box. Within this volume, the location of the lowest favourable energy point based on contributions from the maps of hydrophobic, hydrogen bond acceptor and hydrogen bond donor energies is located and acts as a seed point. Next, in an arbitrarily determined orientation, a search in space, tetrahedral in nature and configurable in granularity is carried out creating further points of favourable high energy. A tetrahedral search is carried out by applying the following vectors to a search distance giving the size of tetrahedrons: (1, 1, 1), (−1, −1, 1), (−1, 1, −1), (1, −1, −1). Applying these vectors to a point as shown in Figure 5.14 generates points distributed in a tetrahedral arrangement around the original point.

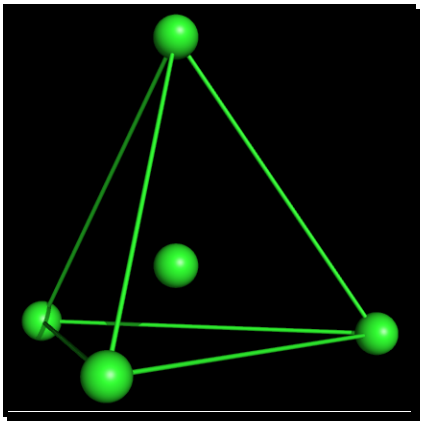


Figure 5.14 - A tetrahedral distribution of points around a given starting point (centre)

These points are extended further out by subsequent tetrahedral searches until non favourable energy is found (determined by user definable cut-off values) and the search in that direction stopped. The program allows outputting of points at any stage as pseudo atoms in the SDF file format. These distributions can then be viewed at intervals in their creation with molecular viewers and the set of points seen

to ‘grow’. With the distribution of points defined, the energy contributions that combined to make the point a favourable high energy point are examined. Depending on simple user definable weightings and the contributions from each map, the point is classified as being a favourable position for hydrophobic, hydrogen bond acceptor or hydrogen bond donor atoms and typed accordingly. With the points generated and typed, this information can then be passed to UFSRAT to generate a set of descriptors describing the space filled by the ideal ligand and the position of chemically favourable atoms.

A number of factors contribute to the program implemented using the above technique not producing distributions of high enough quality for use as input to the UFSRAT process. A major problem is the search algorithm used; ideally, a more intelligent incremental construction type approach would be used where a feasible ligand structure was made, taking into account the proximity of surrounding ligand atoms and realistic intramolecular ligand bonds being made. Furthermore, the stopping criterion was shown to be problematic. Without creating hundreds of candidate distributions and testing each, the search space was deemed too large for the simplistic search method which was implemented. Shown in Figure 5.15 is the protein FKBP12 with the starting seed position identified by a yellow point in step 1. This point in the protein is the most energetically favourable hydrophobic position within the volume. Figure 5.15 also shows the growth of points at 1, 100, 250, 500 and 999 steps. 999 steps due to the visualisation using the SDF file format and a

limitation of the format allowing only between 1 and 999 atoms to be present in a molecule.

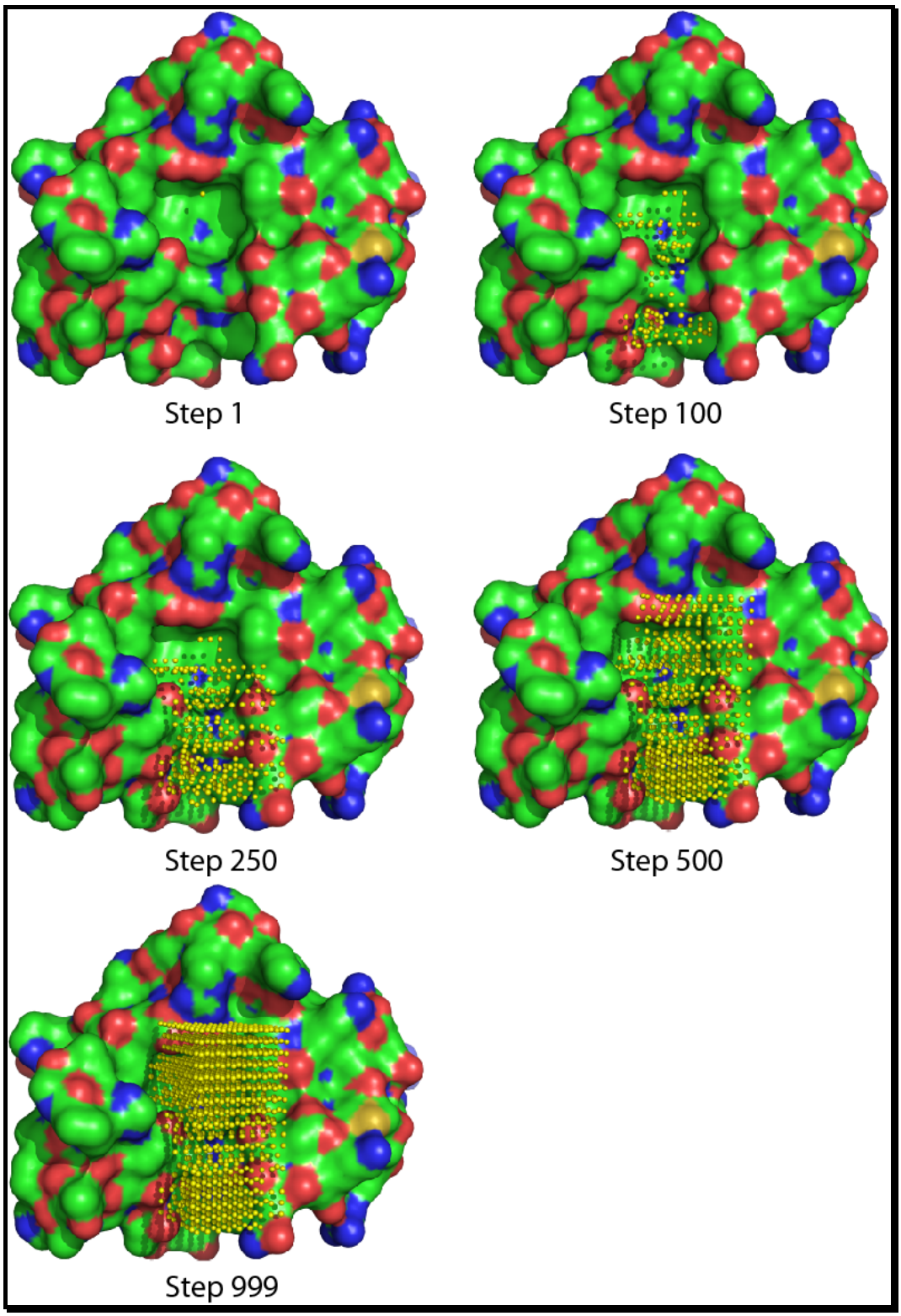


Figure 5.15 - A run of the program mapvolume on the protein FKBP12, bounding box in position of fujimycin. Placement of points shown at steps 1, 100, 250, 500 and 999

An interesting result was observed during development and testing of the program. When run without a bounding box on the same protein and structural data as above, the program starts at the lowest energetically favourable point. It was assumed that this would be located in the binding site that the natural ligand FK-506 or fujimycin binds. This was found not to be the case and a point on the opposite side of the protein was determined to posses the lowest energy. The point lies in a small inset surrounded on nearly all sides and thus represents a deeply buried binding site, which accounts for the low energy. Figure 5.16 shows this point. As the algorithm runs, a low energy path is made and points placed along it on the surface of the protein which leads to the point defining algorithm reaching the familiar binding pocket of fujimycin. Steps 1 and 10 show FKBP12 from the opposite side as the fujimycin pocket. Step 100 shows a side view allowing the path from the lowest energy point to the known binding site. Step 250 shows the point placing algorithm has filled the pocket and gone on to other low energy points. It is able to do this as no bounding box is present.

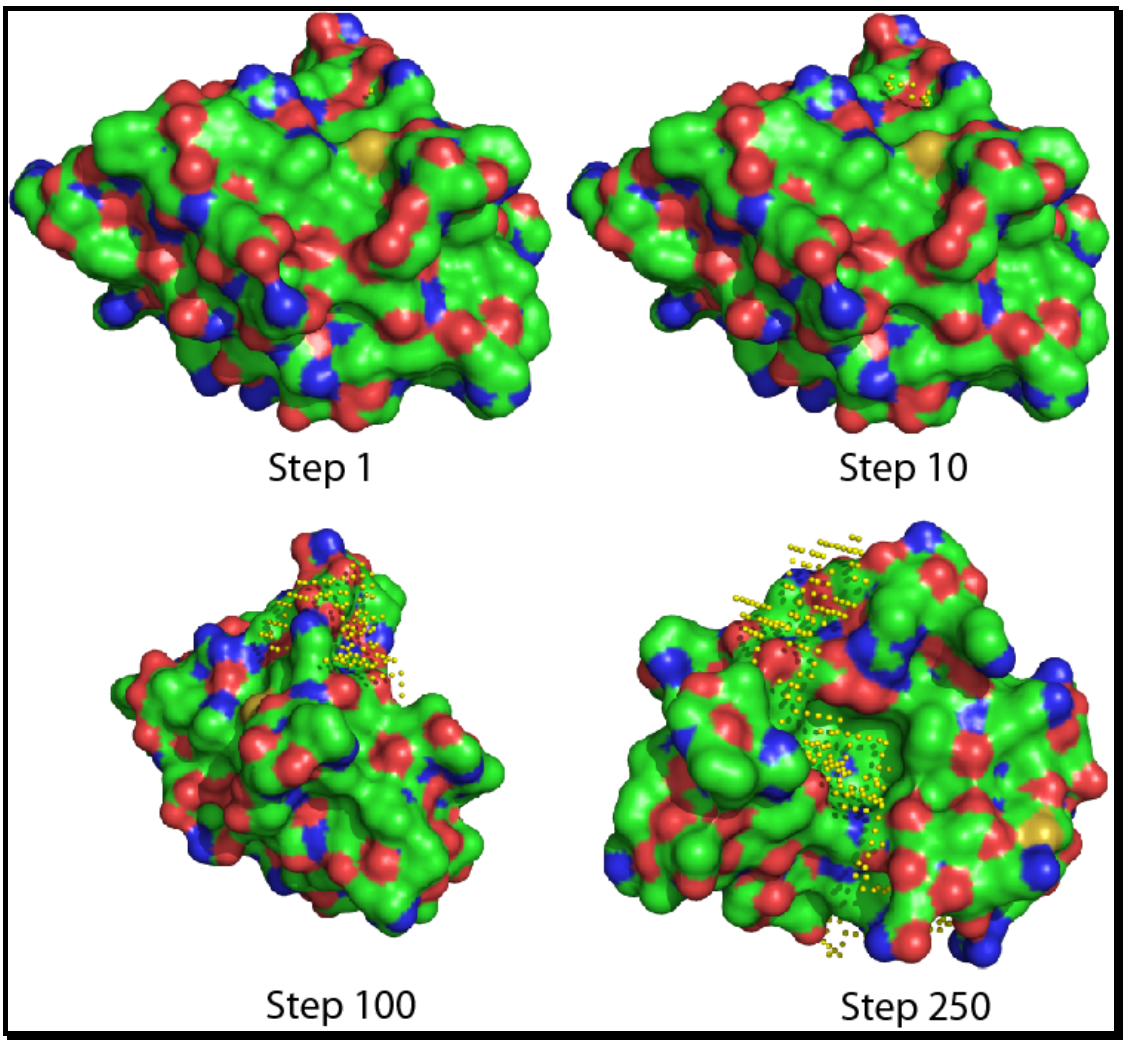


Figure 5.16 - A run of the program mapvolume on the protein FKBP12 without a bounding box restricting the search. Placement of points shown at steps 1, 10, 100 and 250

As mentioned previously, the point distributions were not of significant enough quality to be used for UFSRAT descriptor calculation. The ideal output from this program would have been small distributions which took on rough approximations to small molecules. It was hoped that this would be the case and that the tetrahedral search pattern would approximate this well. However, as no stopping criteria are present, the algorithm continues until all of the high energy space has been filled. This is particularly evident in Figure 5.15 which shows all available space in the

boundary box being filled. A solution to this problem could be to limit the algorithm to a certain number of steps, twenty to fifty for example. However, the paths generated are thin and single layer in nature as shown by Figure 5.16. The program may have some value in suggesting possible points to a user who then goes on to craft a distribution whilst looking at the features present in the protein. This is far from the automated ideal ligand descriptor generator initially envisaged.

5.3.5 Prediction and validation of 11-beta HSD1 inhibitors using UFSRAT

11β-hydroxysteroid dehydrogenase (11ßHSD1) is an important protein that converts the glucocorticoid cortisone to cortisol. Over-production of cortisol has been linked to many health issues such as obesity and diabetes in mice and humans [23]. Glucocorticoids in general display links to obesity, diabetes, cardiovascular disease and also has an impact on memory function [25]. Controlling the rate at which cortisone is turned over into cortisol would be advantageous and this makes 11ßHSD1 a drug target of great importance. With a host of inhibitors already known for the protein, UFSRAT was used in an attempt to find similars to the known inhibitors. Four query structures were selected; carbenoxolone, 2-anilinothiazolone, cortisol and an adamantane. A search of the EDULISS database was then carried out for each of the queries, saving the top 500 similars for each. Lipinski's rule of 5 was applied to the lists and also LogP filtering to help ensure solubility of the suggested similars. Manual inspection was then used to remove unfavourable and toxic molecules. A top 10 to 20 molecules were then selected from each set of filtered

query results. From a total of 50 selected molecules, the relevant suppliers of the molecules were contacted and an attempt at purchase made. At the time of buying, only 34 of the molecules were available for purchase. The compounds were then tested in a biochemical laboratory by Jillian Adie, a co-worker within the department. A selection of assays and techniques on 11ßHSD1 from human and mouse expressing the recombinant protein through yeast and also cell cultures shows success and inhibition has been achieved. Scintillation proximity assay (SPA) and fluorescence assays on the protein in different forms across mouse and human 11ßHSD1 shows that of the 34 purchased, 24 compounds show inhibition in at least one target protein and one assay.

Query molecule	Number purchased	>30% inhibition of cortisone to cortisol turnover
Carbenoxolone	16	6
2-anilinothiazolone	14	3
Adamantane	4	1

SPA resulted in 10 compounds exhibiting >30% inhibition.

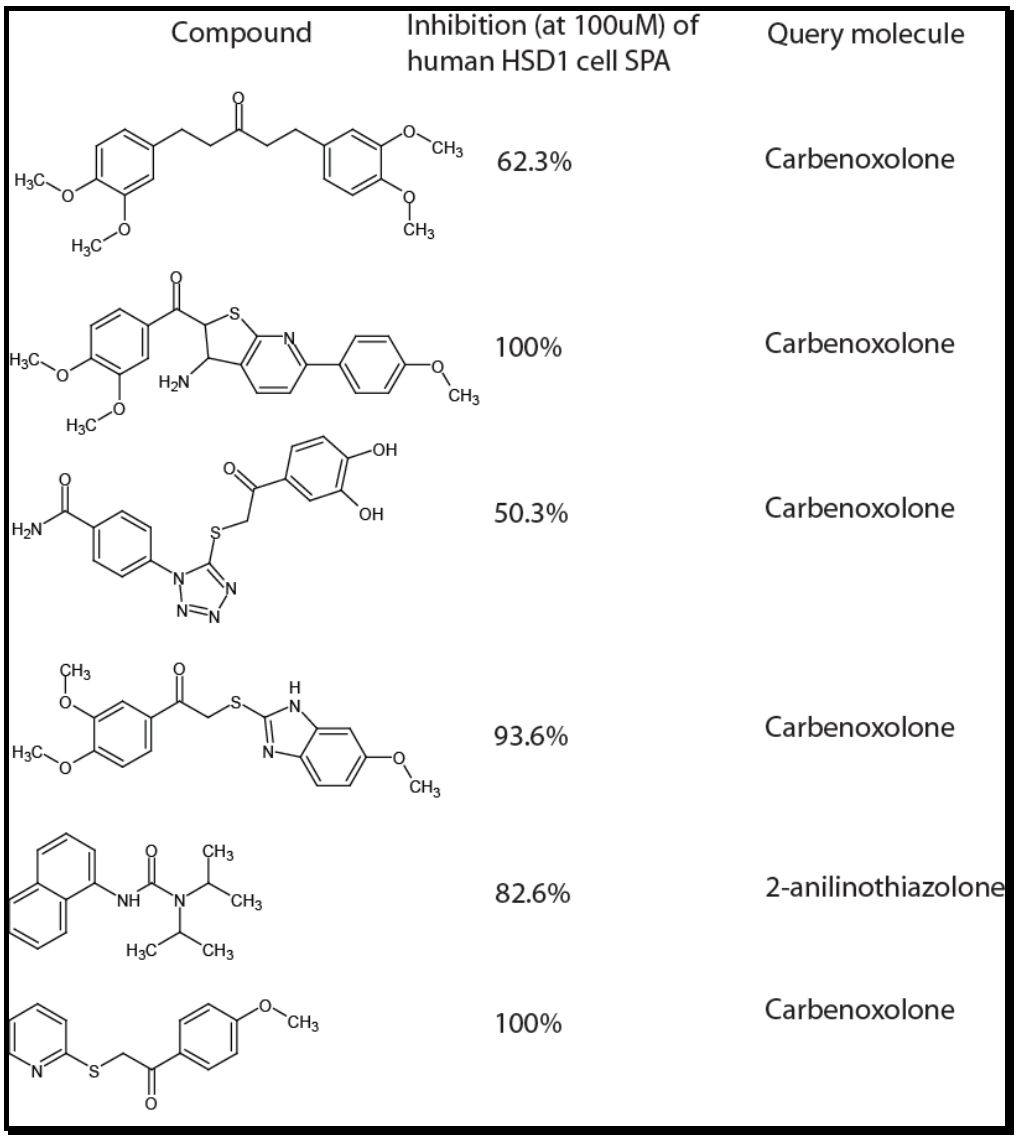


Figure 5.17 – Top 11ßHSD1 inhibitors as confirmed using SPA

Results from recombinant enzyme fluorescence assay added to our list 3 actives not previously picked using cell SPA. At the time of writing, biochemical lab results are still forthcoming and further analysis and refinement of techniques is required to take these hits forward. Early laboratory results from this exercise using UFSRAT have been extremely successful and show that UFSRAT is able to select compounds that make similar biochemical interactions as the query molecule.

5.3.6 Prediction and validation of FKBP12 ligands using UFSRAT

A dataset containing results from a high throughput NMR screen of small molecules against FKBP12 is available from the San Diego Centre for Chemical Genomics. In collaboration with Liz Blackburn, two small molecule binders from the San Diego dataset were targeted and UFSRAT runs carried out, screening the EDULISS database against each. Alongside this, a small 'core' or rapamycin (natural ligand) was investigated.

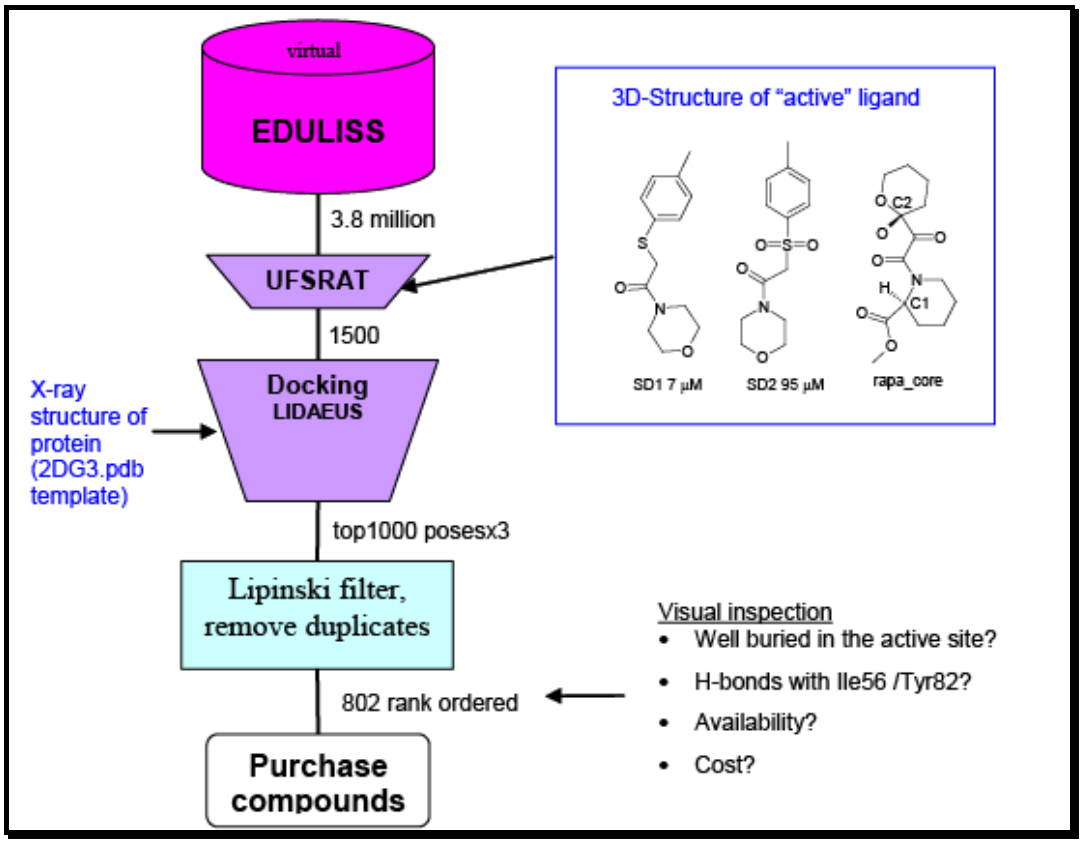


Figure 5.18 - The selection process undertaken in constructing the SL dataset. Reproduced from Biophysical Studies of Protein-ligand Interactions and the Discovery of FKBP12 Inhibitors, Thesis authored by Liz Blackburn Submitted 2009

Purchasing of the compounds created a series named SL (Steven Shave/Liz Blackburn), upon which a thermal denaturation fluorescence assay was used by Liz Blackburn in the screening process to detect binding.

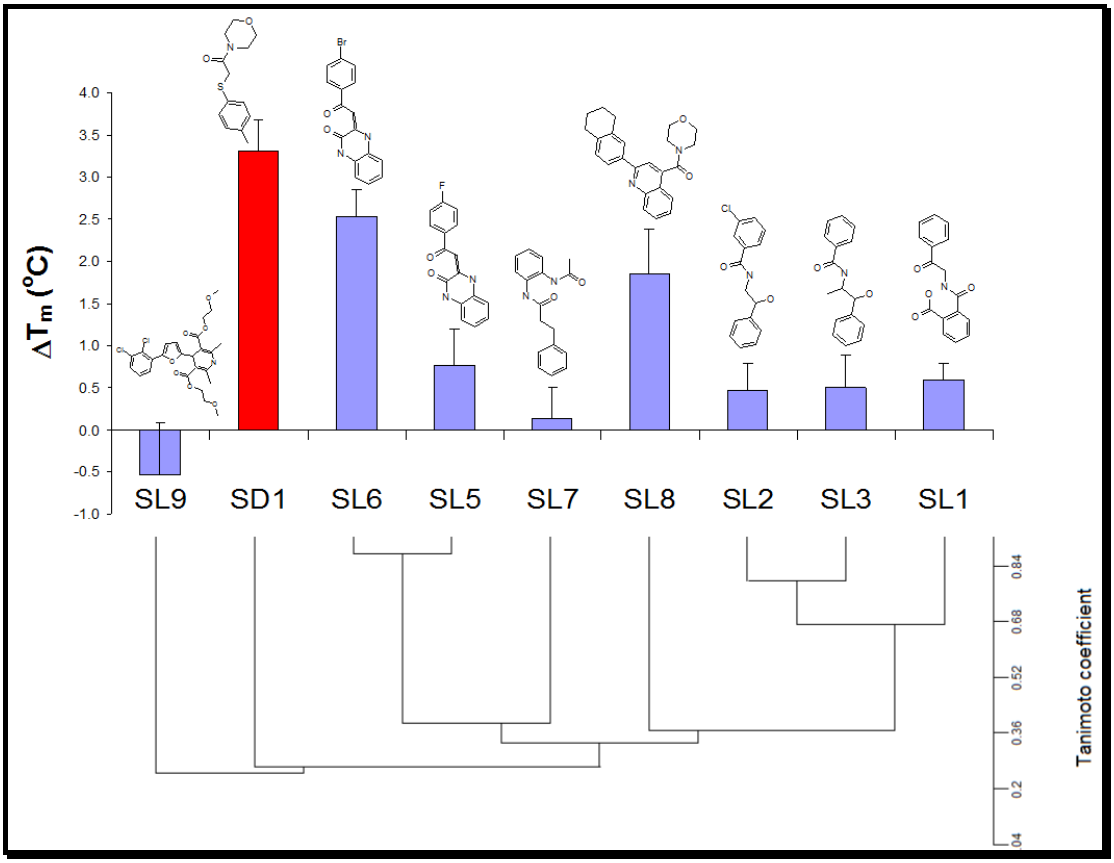


Figure 5.19 - Thermal denaturation fluorescence assay results for key members of the SL series. Taken from the PhD thesis of Liz Blackburn [42]

Relating to Figure 5.19, from "Biophysical Studies of Protein-ligand Interactions and the Discovery of FKBP12 Inhibitors [42]:

"TDF assay results for the SL series. 10μM FKBP12, 250μM ligand/SD1; 10x SYPRO orange, 1 % MeOH, 0.83 % DMSO, 30mM ammonium acetate; 40 mM Bis-Tris, pH 7. Standard deviation is shown for n = 3 (three separate experiments). The cluster dendrogram is shown to illustrate similar compounds."

SL8 and SL6 as shown above in Figure 5.19 were deemed the strongest binders. SD1, the strongest binder from the San Diego dataset is shown for reference, increasing the melting temperature of FKBP12 by nearly 3.5 degrees in the assay. Activity for SL8 was predicted to be around 65 μM and SL6 around 49 μM.

A full write up concerning the SL series is available in Liz Blackburn's thesis "Studies of protein-ligand interactions and the discovery of new FKBP12 inhibitors".

5.4 Volumetric Shape Recognition with Atom Types (VolRat)

VolRAT is an extension of UFSRAT. However, VolRAT uses a comparatively more densely packed set of points as input to its descriptor calculation routines. Essentially, the volume occupied by the molecule and dictated by its shape is used as the input. To achieve this, the atoms are typed in the standard way using LIDAEUS [107; 127; 180] routines. Then, taking each atom in turn, it is evaluated according to the predicted properties which may be of type hydrophobic, hydrogen bond acceptor or hydrogen bond donor. The program then carries out a search of the space surrounding the atom up to a user definable limit and fills it with points of user definable spacing. Figure 5.20 illustrates the volume defined by points being generated for a protected tryptophan molecule.

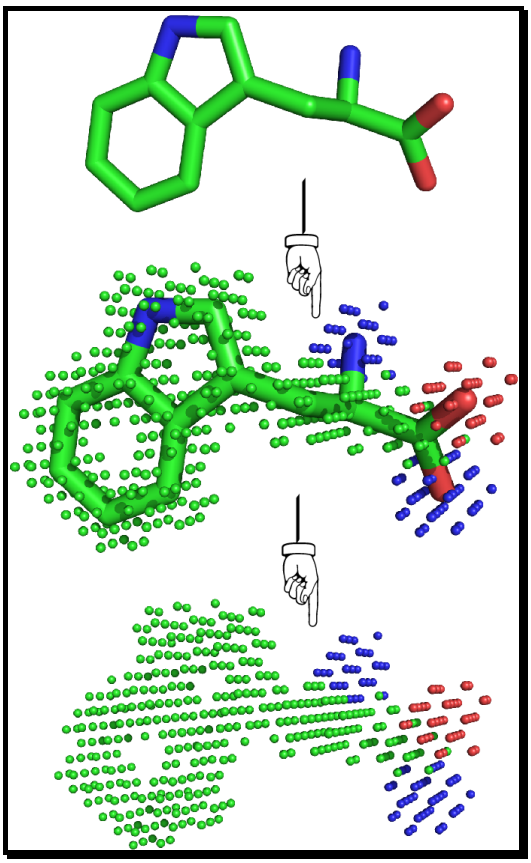


Figure 5.20 - VolRAT point calculation using protected tryptophan with points plotted at a radius of 0.5 Å around atoms and a grid spacing of 0.25 Å. The oxygen atoms of the carboxylic group have both hydrogen bond acceptor and donor properties and therefore points for each atom are included in each distribution. Limitations in the visualisation program only allow the display of one property for each oxygen atom.

Using these new points derived from nearby atoms, the positions are used as input to a standard UFSRAT run. The perceived value in this approach lies in the idea that having a slightly blurred distribution describing the molecule can lead to different chemical features producing similar space filling patterns. These interchangeable chemical features allow greater chemical diversity whilst maintaining molecular similarity as seen by UFSRAT based techniques. We may reasonably expect in certain situations to achieve greater diversity in the top rated similars.

5.4.1 VolRAT pseudo code

```
Read in query molecule

Type atoms using LIDAEUS typing system

Create fake point set, tetrahedral filling of space within atom VDW radii.

Create four distributions containing points for all atoms (and fake points) for
hydrophobic, hydrogen bond acceptor and hydrogen bond donor points.

Calculate point positions

Calculate UFSR descriptors for each of these four distributions, giving (4 x 12) 48
descriptors which make up a set of UFSRAT descriptors.

While candidate molecules can still be read in:

        Read in candidate molecule.

        Type atoms using LIDAEUS typing system and create fake pointset as above

        Create four distributions containing points for all fake points,
        hydrophobic, hydrogen bond acceptor and hydrogen bond donor points.

        Calculate UFSR descriptors for each of these four distributions, giving (4
        x 12) 48 descriptors which make up a set of UFSRAT descriptors.

        For each value in 4 descriptor sets of 12 UFSR distributions, find
        difference between query and candidate and sum, divide by 12 and apply
        appropriate weighting the cumulative contribution from the 4 distributions
        is the score.

        Add one to the score

        Take the reciprocal of the score to achieve a UFSR score

        Add SIMILARITY key to query molecule documenting the score achieved

        If score is good enough to be kept in the top X, then add the molecule in
        the correct position to the list to be kept

        If the list to be kept is too big, remove the lowest scoring molecule from
        the list.

End while

Loop over list to be kept outputting suitable candidate molecules
```

5.4.2 Common Chemicals and their VolRAT predicted similars

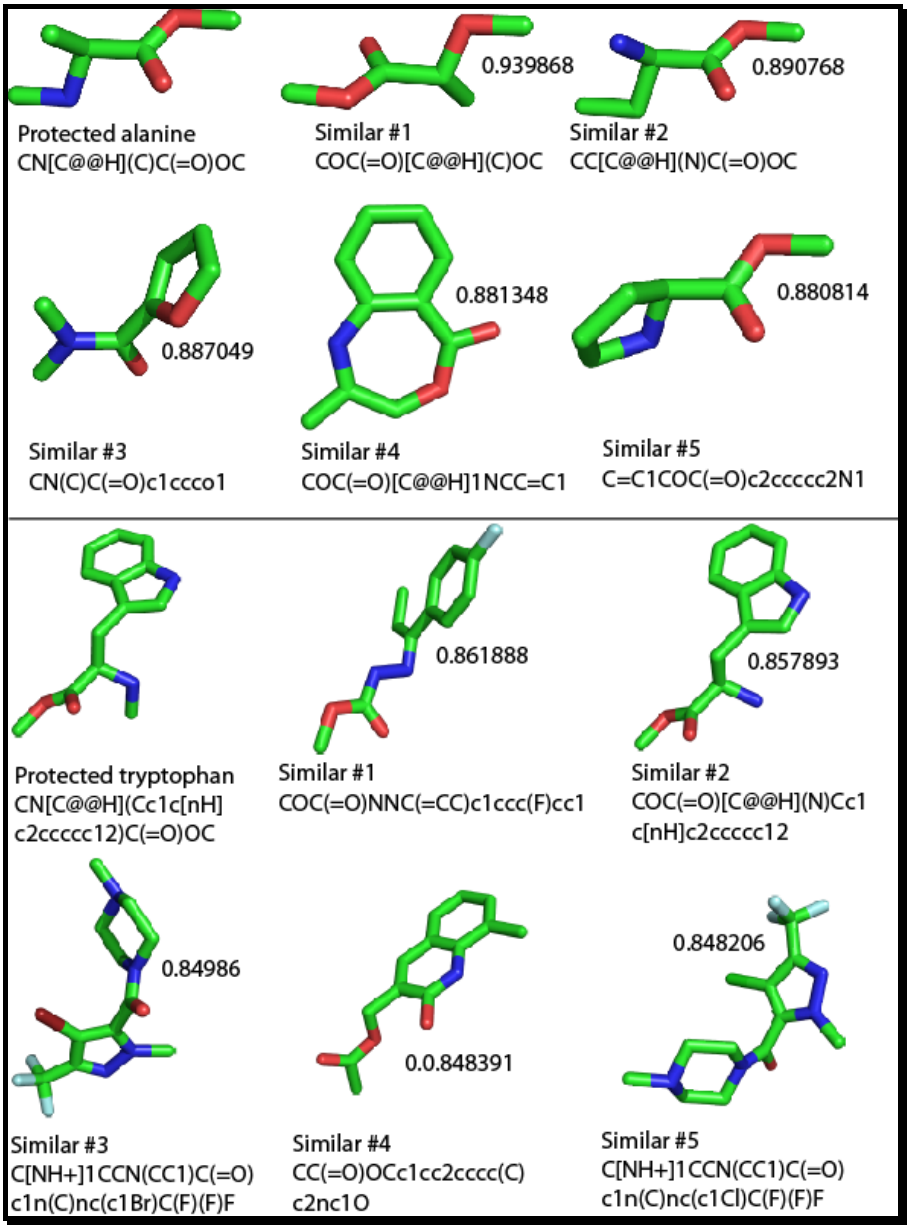


Figure 5.21 – Protected alanine and protected tryptophan VolRAT predicted similars with scores and SMILES representations. Molecules scored and similars searched from EDULISS dataset of 3.67 million molecules

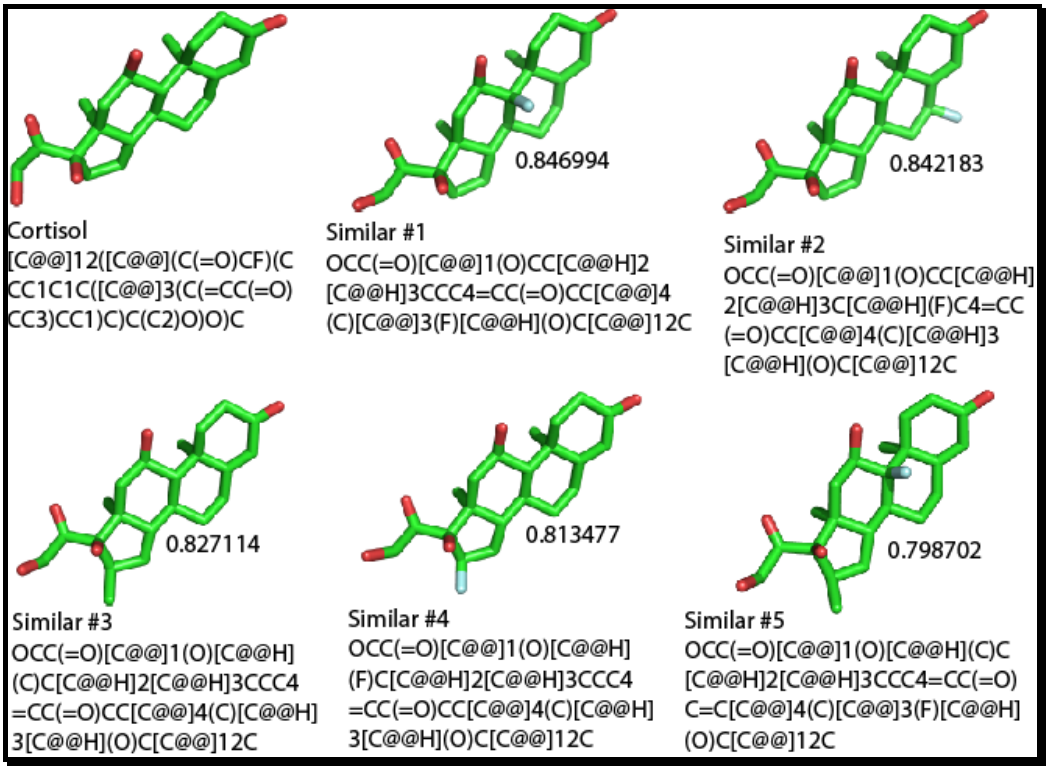


Figure 5.22 - Cortisol UFSR predicted similars with scores and SMILES representations

Subjective visual inspection of the VolRAT similars in Figure 5.21 and Figure 5.22 shows promising results. The similars sharing a lot in common with the UFSRAT predicted similars to the common chemicals. Again, the overall shape of the molecule has lessened in importance, this time to a greater degree as evident in protected alanine similar 4.

5.4.3 VolRAT profiling

VolRAT profiling has been carried out in the same way as in section 5.2.3 using the same supplier catalogues from Specs and Sigma and the same hardware.

Execution time with the Specs catalogue providing the candidate molecules was 14 hours, 9 minutes and 25 seconds. This equates to roughly 4.4 candidate-query comparisons per second. 1 million comparisons are therefore predicted to take ~62.5 hours.

Execution time with the Sigma catalogue providing the candidate molecules was 10 hours, 15 minutes and 44 seconds. This equates to roughly 5.4 candidate-query comparisons per second. 1 million comparisons are therefore predicted to take ~51.4 hours.

Similar to UFSR and UFSRAT, VolRAT computation is of the order 4N on atoms (depending on atom types encountered).

As described earlier, runtimes for the Specs catalogue searched using UFSR and UFSRAT was ~85% of the Sigma catalogue. This trend is present to a lesser extent with VolRAT, Specs taking 81.5% of the Sigma catalogue runtime.

The performance exhibited by VolRAT is restrictive, taking well over two days to screen 1 million molecules on the test system. Also, one could predict that the quality of similars produced could be of equal or lesser quality to UFSRAT predictions in most situations.

Figure 5.23 shows a VolRAT run using candidate molecules from the entire EDULISS database (~1.67 million compounds) and shows the score distribution achieved with three query molecules (protected alanine, protected tryptophan and cortisol).

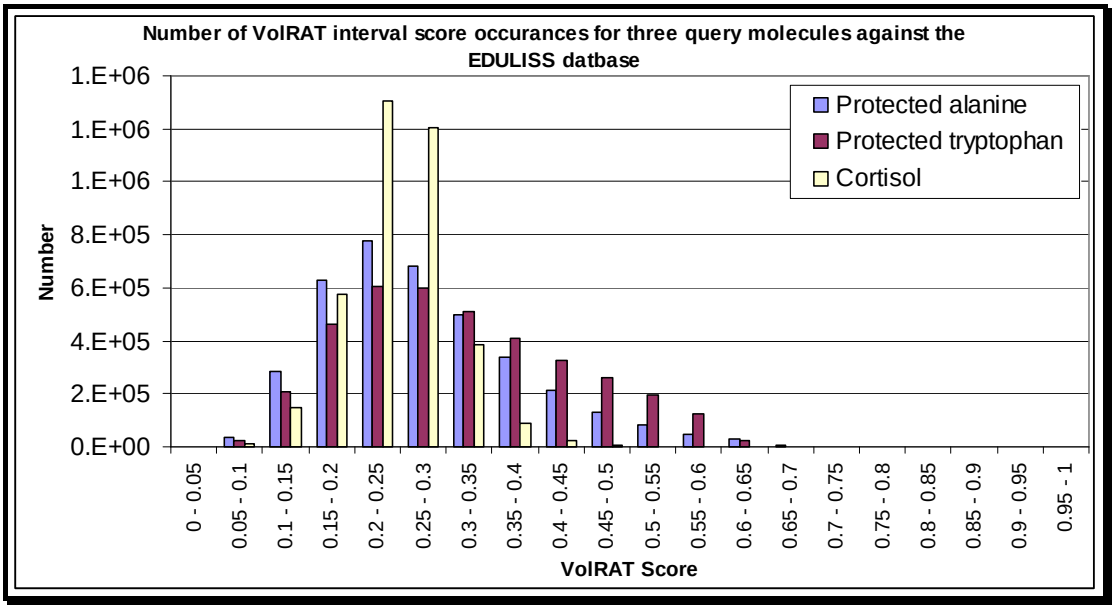


Figure 5.23 - Number of VolRAT interval score occurrences for three query molecules against the EDULISS database. Score distribution and the discriminatory abilities of VolRAT can be seen

5.5 UFSRGraph

A major restriction present in the above techniques (UFSR, UFSRAT and VolRAT) is that the degree to which a match can be made depends upon the conformation of molecules taken as input to the algorithms. This can be a problem especially in situations with large peptides or isomers, such as cis-trans isomers or enantiomers. The candidate molecules all have their conformations generated using concord, and so it is practice to supply the query molecule with its predicted lowest energy vacuum conformation also coming from concord. If however, the query molecule comes from a source which also describes atom positions and overall conformation, such as PDB files solved using x-ray crystallography, the correct 'bound' conformation may well be used. One could imagine a situation where the conformation adopted by a bound ligand is greatly different from its lowest energy vacuum conformation. In this case, the comparison being made will be of a biologically active conformation and predicted vacuum conformations. The aim of this technique was to be able to apply UFSR techniques as built upon by UFSRAT and VolRAT but in a conformation independent manner.

UFSRGraph decomposes molecules into a graph representation based on atom connectivity. The vertices of the graph represent atoms and the edges between atoms, bonds. The edges are also weighted with the ideal bond length for each specific bond made between the typed atoms connected by it. These ideal lengths are taken from the Tripos force field (Chapter 2) and atoms are typed using code taken

from the virtual screening program LIDAEUS. A graph of the protected alanine molecule would be conceptually represented as shown in Figure 5.24.

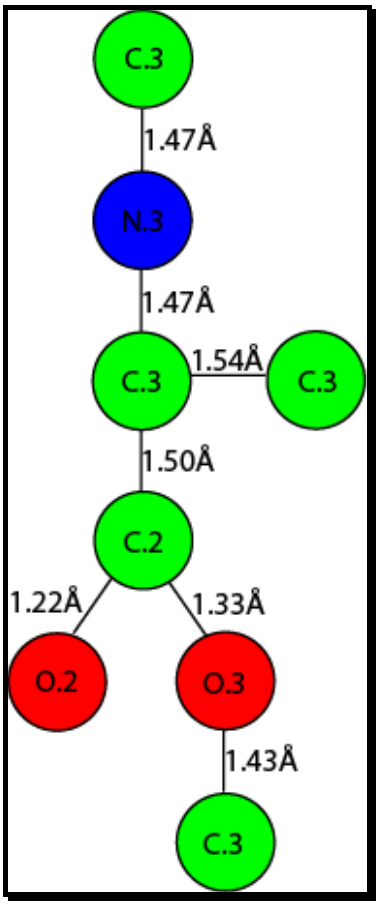


Figure 5.24 - Graph representation of protected alanine with typed atoms represented as nodes and edges weighted by ideal bond length

UFSR like techniques are then applied to the graph to generate the descriptors. However, only three points of interest are defined instead of four as in the UFSR, UFSRAT and VolRAT techniques. With the graph of a molecule constructed based on connectivity and the graph edges weighted by ideal bond length, all possible paths within the molecule are examined and the longest found. The length of this path is

then determined taking a cumulative measure of edge weights. The node (atom) which lies closest to the middle point of this path is defined as Point 1 (P1). A list of shortest distances from all nodes to node P1 is then generated and from this list the mean, variance and skew of the distribution calculated. These serve as descriptors one to three. Point 2 (P2) is then defined as the most distant node from P1. Again, a distribution of node distances is calculated and the mean, variance and skew of the distribution goes to make up descriptors four to six. The final point is the furthest node from P2 and termed Point 3 (P3). The same method of generating a distribution of node distances from this point is used again and the mean, variance and skew go to make descriptors seven to nine.

5.5.1 UFSRGraph pseudo code

```
Read in query molecule
Type atoms using LIDAEUS typing system
Create a graph object of the molecule
Walk graph and save all paths made of unique nodes
Find the longest of all paths
Find and save the shortest path for each atom to every other atom
Define P1 as the node closest to the middle point of the longest path.
From the list of shortest paths, get the distances of all nodes (atoms) to P1
Calculate the mean, variance and skew of this distribution and assign these as
descriptors 1 to 3
From the list of shortest paths, find the node (atom) which is furthest from P1 and
define this as P2.
From shortest paths get the distances of all nodes (atoms) from P2.
Calculate the mean, variance and skew of this distribution and assign these as
descriptors 4 to 6.
From the shortest paths, find the node (atom) which is furthest from P2 and define
this as P3.
From shortest paths get the distances of all nodes (atoms) from P3.
Calculate the mean, variance and skew of this distribution and assign these as
descriptors 7 to 9.

Read in query molecules and perform the same descriptor generation as above
generating 9 descriptors.

For each value of the 9 descriptors, find the difference between query and
candidate and sum, divide by 9
Add one to the score
Take the reciprocal of the score to achieve a UFSR score
Add SIMILARITY key to query molecule documenting the score achieved
If score is good enough to be kept in the top X, then add the molecule in the
correct position to the list to be kept
       If the list to be kept is too big, remove the lowest scoring molecule from
the list.
End while
Loop over list to be kept outputting suitable candidate molecules
```

5.5.2 UFSRGraph Profiling

UFSRGraph profiling has been carried out in the same way as in section 5.2.3 UFSR Profiling using the same supplier catalogues from Specs and Sigma and the same hardware.

Execution time with the Specs catalogue providing the candidate molecules was 58 hours, 30 minutes and 42 seconds. This equates to roughly 1.1 candidate-query comparisons per second. 1 million comparisons are therefore predicted to take ~258.5 hours.

Execution time with the Sigma catalogue providing the candidate molecules was 53 hours, 48 minutes and 27 seconds. This equates to roughly 1.03 candidate-query comparisons per second. 1 million comparisons are therefore predicted to take ~269.7 hours.

Observing the figures above, the usefulness of UFSRGraph is called into question. The massive amount of time required to screen a large number of molecules is restrictive. However, performing the profiling run with calculation time for the descriptors of each molecule shows that most molecules are profiled very quickly, in the region of 0.01 seconds in many cases for small drug like molecules. The overall performance is harmed greatly by molecules which can be defined as problem

molecules. Take for example the molecule shown in Figure 5.25. This molecule from the Sigma dataset has proved the most computationally expensive, UFSRGraph taking 2 hours 23 minutes 55 seconds on modest hardware. This is down to the number of rings present in the molecule. During descriptor calculation, all possible paths from each atom to every other atom are explored. If atoms at either end of the molecule are explored, then the number of paths increases exponentially with the number of rings encountered as a ring presents two possibilities. A large fused ring system exacerbates the problem to a greater degree.

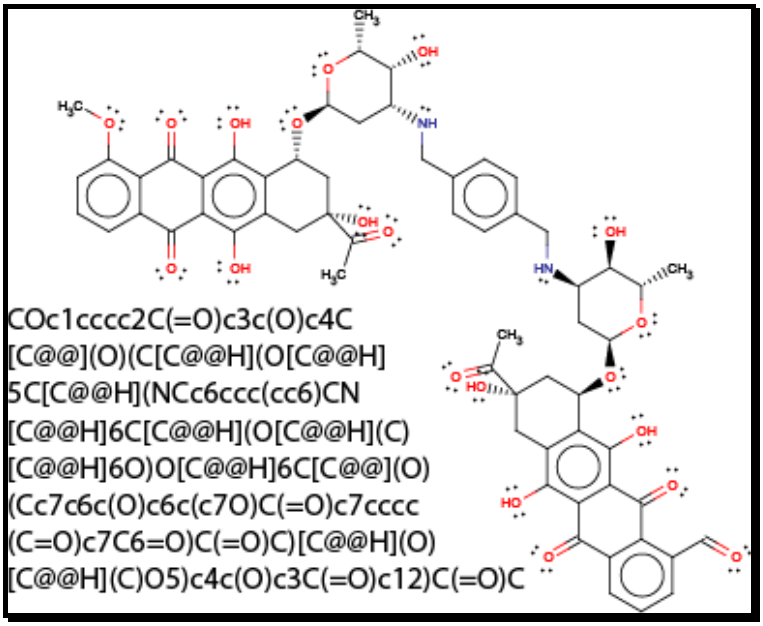


Figure 5.25 - A problematic molecule for UFSRGraph descriptor calculation

Figure 5.26 shows a UFSRGraph run using candidate molecules from the entire EDULISS database (~1.67 million compounds) and shows the score distribution achieved with three query molecules (protected alanine, protected tryptophan and cortisol).

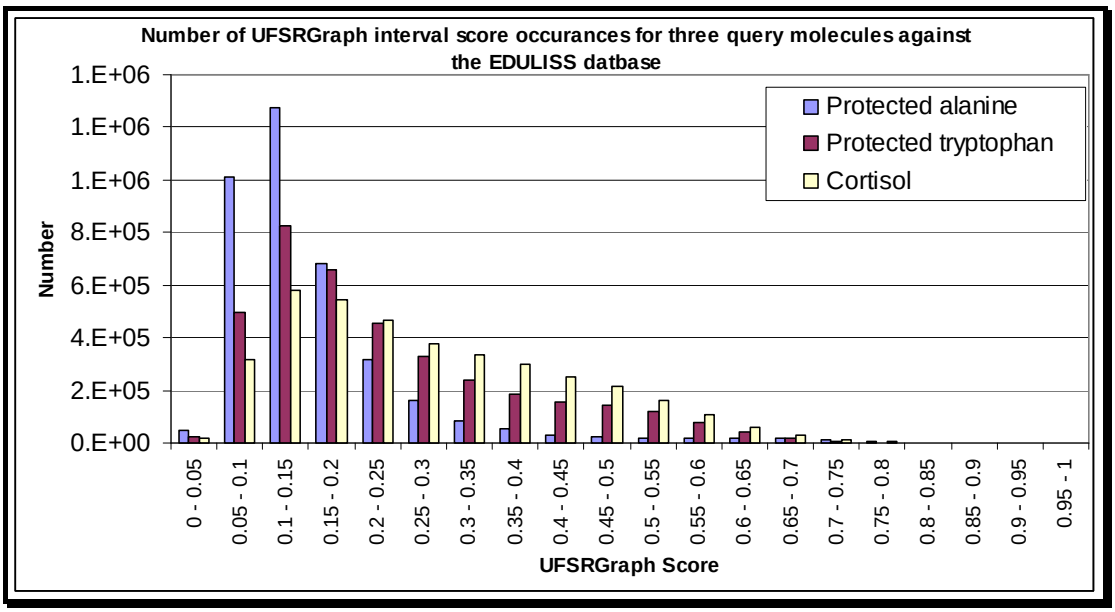


Figure 5.26 - Number of UFSRGraph interval score occurrences for three query molecules against the EDULISS database. Score distribution and the discriminatory abilities of UFSRGraph can be seen

5.5.3 Common chemicals and their UFSRGraph Similars

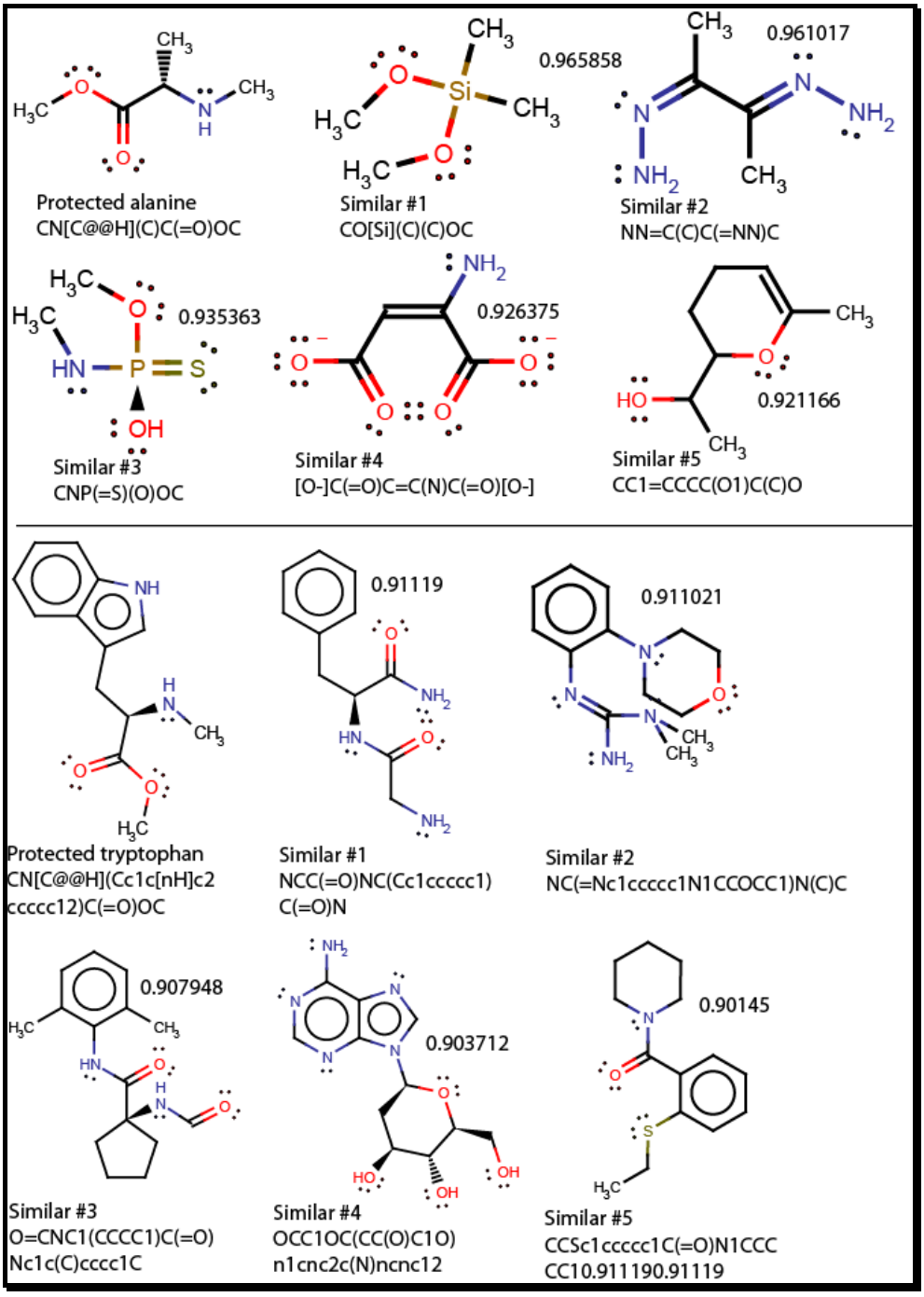


Figure 5.27 – Protected alanine and protected tryptophan UFSRGraph predicted similars with scores and SMILES representations. Molecules scored and similars searched from EDULISS dataset of 3.67 million molecules

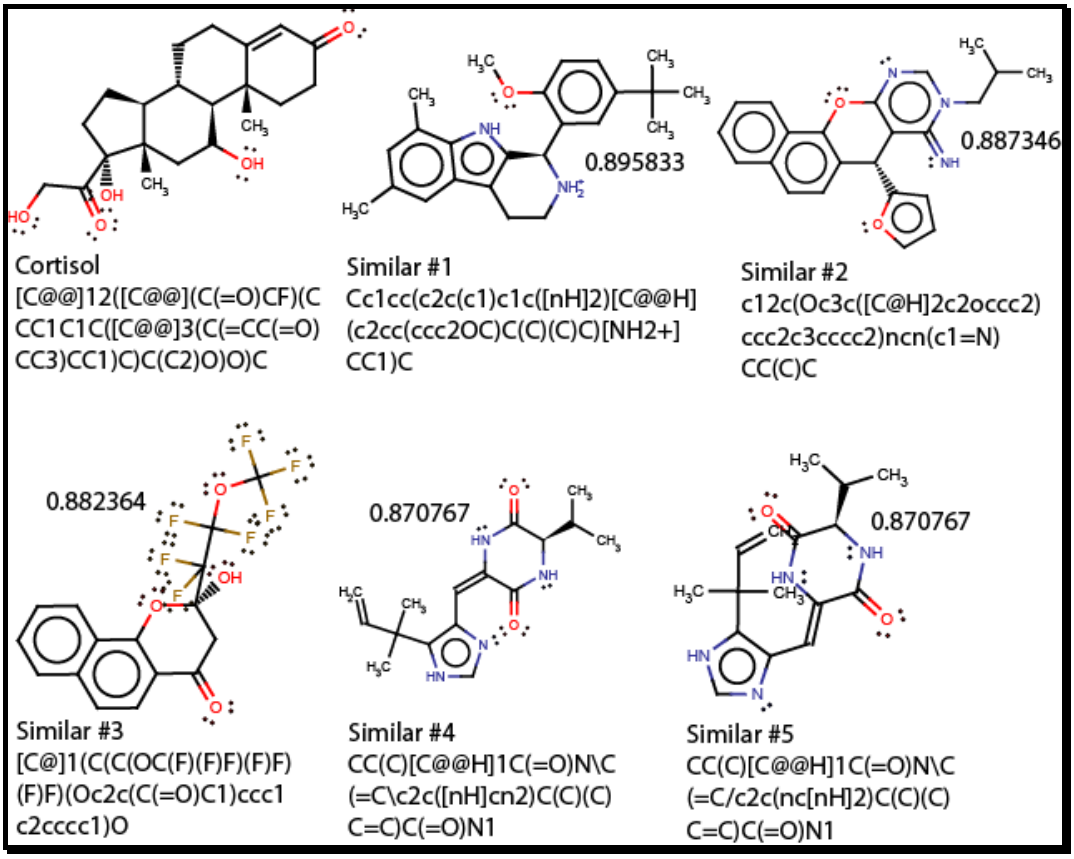


Figure 5.28 - Cortisol UFSR predicted similars with scores and SMILES representations.

The UFSRGraph common chemical predicted similars in Figure 5.27 and Figure 5.28 are interesting in that the graph of the molecules produced appears roughly similar but the chemical structures present differ dramatically.

5.6 Pre-calculation of descriptors for near instant results

As the UFSR based techniques mentioned above are used within the Centre for Translational and Chemical Biology against an existing database (the EDULISS database), the opportunity existed to dramatically decrease the amount of time taken to perform a UFSR based search of the database. With descriptors for each flavour of the technique (UFSRAT, VolRAT and UFSRGraph) pre-generated and stored in an easily addressable format, only calculation of query molecule descriptors is required. Performing a dramatically sped up UFSRAT run takes the following form: i) user supplies query molecule and its UFSRAT descriptors are generated. ii) The pre-generated descriptors for every molecule in the EDULISS database are retrieved and scored using the standard scoring function. iii) A list of the top scoring candidate molecules is created by the accompanying unique identification number carried with the descriptor information. Upon completion, the program outputs a sorted list containing the unique identifier of the predicted matches along with their score. This list of unique identifiers can then be used to query the EDULISS database and a dataset created. Another approach to obtaining the molecules from their unique identifiers is to use the program extractmols, to which the list of unique identifiers and their associated similarity score, can be passed. This program will then read through a large SDF file (containing all molecules within the EDULISS database) finding the molecules of interest, appending a similarity key containing the name of the query molecule and the similarity score between the query and candidate. These extracted molecules can then be written to an SDF file. One drawback of having the descriptors of molecules pre-generated is that, upon the

addition of new molecules to the EDULISS database, they must have their descriptors generated and included into the resource used by these programs. The chosen method of storing these pre-generated descriptors takes the form of a custom designed binary file and associated format. The choice was made not to include this information directly within the EDULISS database as the amount of overhead computation and input/output would have been too great within a database environment. The binary file representing UFSRAT descriptors for every molecule in the EDULISS database is created using a number of programs. The first stage is descriptor calculation, which is dealt with by the program "descscorebinary". With a large SDF file containing the EDULISS database named eduliss.sdf, the program can be invoked in the following way:

```
./ufsratgenbin monolith.dat edulissUFSRATdesc.bin < eduliss.sdf
```

The program takes as an argument, the monolith.dat file. This is the monolith.dat file from a standard LIDAEUS run. This essentially describes the different atoms types and their properties so that UFSRAT knows which distribution atom information within the molecule should be stored within. The second argument is the filename for the binary file containing descriptors and unique identifiers for the processed molecules 'edulissUFSRATdesc.bin'. 'eduliss.sdf' is then piped to the program; this is the SDF file containing all entries within the EDULISS database. When run in this way, the program steps through the input SDF file one molecule at a time, generating UFSRAT descriptors for each entry.

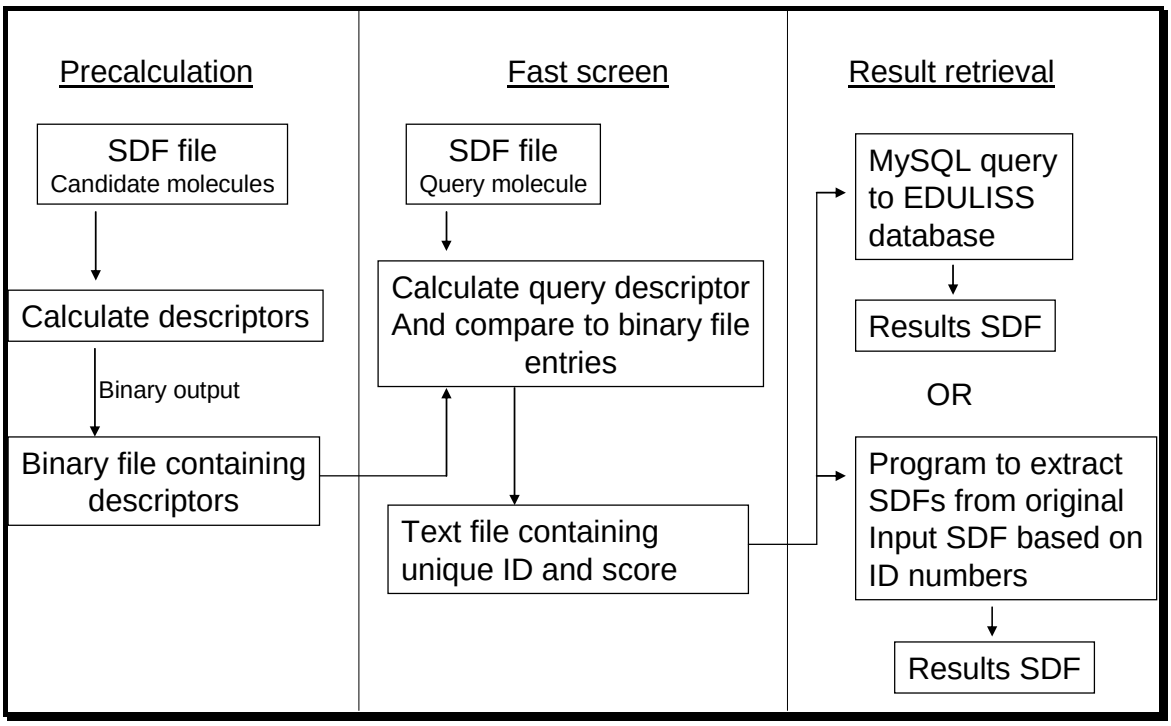


Figure 5.29 - Flow diagram showing conceptually how the pre-calculation of descriptors changes the operation of the UFSR based techniques

An entry within the binary file contains the following information and requires the amount of memory specified below:

- Unique identifier – 20 bytes.
- Number of bytes used in identifier – 4 bytes.
- 48 descriptor values stored as floats – 192 bytes.

Storing all required information for a molecule therefore requires 216 bytes. The file is laid out in the order shown above, the representation of the first molecule starting at byte 1, the second molecule at byte 217 and so on.

The decision to store the data in a custom binary file was taken primarily for the reasons of speed and efficiency. The data could have been operated upon as a human

readable textual representation. However, the same amount of data can be represented in a much smaller amount of memory when encoded as a binary representation of the floating point numbers. If we take for example the floating point number -0.0521045 as located in the descriptor set of a molecule from the EDULISS database, the number represented within a text file would be as a series of characters, each taking 1 byte. The textual representation of the above number takes 10 bytes (including the sign and decimal place as characters). For this number to be represented in the floating point format, only 4 bytes would be required. Additionally, a method of separating the values would need to be present. This would normally be achieved through the use of the tab character. There is no such requirement when using a binary format as the program reading the binary file will have express knowledge of which bytes belong to which data within the binary file. Using a text file to represent the molecules and their descriptors, there will be 50 tab characters per line plus a new line character at the end, 51 wasted characters, and 51 wasted bytes per line (per molecule). In a file containing descriptors for 1 million molecules, the amount of memory wasted in formatting would be just below 50 megabytes ((51*1,000,000)/1,024). In modern computing terms, 50 megabytes per million molecules is not a concern and this alone would not necessitate the conversion into a binary format. The main argument against operation on the textual representation is one of speed. A binary file can be read in, the program knowing explicitly which bytes belong to different data and the bytes copied straight into the computers memory representing a number or other object. Using the textual representation, the file would have to be searched for the delimiting tab character, the text before it being read in and then translated or cast into the data of type it

represents. This parsing and casting is not present (to such a detrimental extent) when dealing with binary data.

With the descriptors pre-generated and existing in a single binary file, a high speed version of UFSRAT can be used. The program ‘ufsratbinaryreader’ is used for this purpose. A run looking for cortisol similars within the EDULISS database using UFSRAT as the similarity metric would be invoked as follows:

```
./ufsrat monolith.dat edulissUFSRATdesc.bin cortisol.sdf 200 > ufsratCortisolVsEduliss.txt
```

As above, the program is invoked with a number of arguments, the first being ‘monolith.dat’, a file taken from and used in standard LIDAEUS runs which helps define atom types and the UFSRAT distributions each belongs to. The second is the binary file containing references to all molecules within the EDULISS database and unique identifiers. As argument three, the query molecule is given in a file of type SDF. The fourth and final parameter is the sort size, or how many molecules the final list of top matches should contain. The output is piped as denoted by ‘> ufsratCortisolVsEduliss.txt’ to a text file. This human readable file contains an ordered list, the highest predicted similar at the top of the file represented by its unique identifier, a comma, and then the similarity score as assigned by UFSRAT. This can then be used to extract the molecule from either the EDULISS database directly or the previously mentioned program ‘extractmols’.

This approach results in a calculation time of ~8 seconds to query a molecule against the whole database of 3.67 million candidate molecules. Previous run times for UFSRAT against the same database of 3.67 million molecules have been around 4 hours 2 minutes.

The above approach of pre-calculating descriptors for candidate molecule has been applied to all UFSR based techniques. The exact same process is taken with the original UFSR. VolRAT and UFSRGraph contain an additional step. Due to the long runtimes present in generating the descriptors for these techniques, the binary file is not generated directly from one program. The descriptors are written out in a human readable text file. This allows the computation of the descriptors to be broken up and run in parallel on multiple machines. Simply joining the text files together after each part has been operated on, and all molecules in the dataset have their descriptors calculated, produces a similar to the binary data file. The text file is then read in by a program which produces the final binary file representing all pre-generated candidate molecule identifiers and descriptors. The descriptor generation was implemented in this was solely to allow the computation to be broken up amongst machines and achieve the computation in a shorter amount of time.

For all approaches, the descriptors for all molecules within the EDULISS database have been calculated. The resultant runtimes for all methods except UFSRGraph are below 15 seconds. The runtime of UFSRGraph depends highly on the query molecule. As described earlier, the amount of computation in UFSRGraph descriptor generation depends highly on the complexity of the graph representing the molecule. It is entirely possible that the fast pre-generated descriptor version of UFSRGraph could be invoked with a complex query molecule which itself takes 10 minutes to have its graph explored and descriptors generated. After this query descriptor generation process, the query against all EDULISS candidate molecule can be carried out in ~9 seconds.

Pre-calculation of UFSR based descriptors has allowed many more runs to be carried out than previously possible. This already fast similarity technique has been improved and almost instant results are obtained in the majority of cases.

6 Summary, conclusions and future work

The research documented in this thesis has been fundamentally involved in finding novel binders and therefore facilitating the discovery of new protein-ligand complexes. The report clearly splits virtual screening into two different approaches; structure based and ligand based. Whilst structure based is the more widely accepted approach, arguably giving the researcher more information to work with, the investigation showed success in both.

A major advance in speeding up throughput has been the parallelisation of LIDAEUS. The main scientific endeavour here was to scale the implementation to an architecture as unique as the Bluegene/l. The parallel sorting algorithm developed performed very well and parallel efficiency across a wide range of processors shows that the parallelisation was a success. The opportunity to develop code for the Bluegene/l has also been invaluable, it being the first in Europe. At the time of deployment, the machine offered unparalleled access to processors and capacity previously out of reach of biological sciences.

Turning to ligand based virtual screening, the technique USFR has been expanded upon for the development of UFSRAT. The new technique tailors the principle to the field of biochemistry, taking into account important features of ligands that were not previously included or considered when calculating similarity. UFSRAT has been validated through work carried out by colleagues in a biochemical laboratory,

finding a number of inhibitors for 11ßHSD1 and FKBP12 binders. Their activity was confirmed using a number of techniques. The further extensions to UFSR and UFSRAT have not been experimentally confirmed due to time constraints. Inspection of the predicted similars shows promise in the techniques and shows that their implementation has been successful even though the theory has not yet been validated.

The two main achievements discussed above have been the result of a lot of background and exploratory programming. The early development of the SDF toolkit documented in Chapter 3 has been invaluable during the course of the research, which focussed on using the SDF file format for representation of almost all chemical data. This standardisation saved a lot of time and effort involved in the conversion of formats. The application which allows user-guided docking using LIDAEUS was also another success, although the rendering quality was poor compared to other available programs. The development of a 3D application such as this allowed the author to get to grips with the overall level of involvement required for the representation and rendering of chemical information in 3D.

The attempt to incorporate flexible ligand docking into LIDAEUS was not successful. The approach was possibly flawed on two levels. The force field implemented was too primitive and did not take into account enough terms in its energy calculations. This meant that the energy profiles explored by the minimiser

were too simplistic and led to unrealistic conformations. Fixing the non planar ring problems by the grouping of ring atoms into islands possibly proved too challenging for the minimiser, which upon attempting to move one atom to a lower energy position, ended up taking into account the effects experienced by up to six atoms. The chances of making a favourable movement on one atom are much greater than making the same move on six and expecting each to be beneficial. The second point at which the approach failed was that it proved to be too costly computationally. The property of LIDAEUS that enables it to compete with other codes which are performing flexible docking is that the trade off is in the speed. LIDAEUS has always been designed to be high throughput: the testing of a ligand (i.e. the prediction of all its LIDAEUS-generated complexes) has to take around one second to perform on a modern CPU.

There are many parts of the research carried out that provide starting points for further work. It would be advantageous to develop a force field to enable flexible docking within LIDAEUS, even though this would inevitably remove the previously stuck by golden rule that LIDAEUS must be extremely quick. This would be a strong tool to have, possibly offering the ability to aid in the solving of crystal structures, by knowing the exact conformation and position of the ligand before solving and refinement commences. This rather ambitious goal would not be a small project and would require great care and attention to detail each step of the way. A more achievable goal would be to offer LIDAEUS as a service. Possibly accessible through a web interface, abstracting away all of the parallel code, a job could be

submitted by a researcher, sent off to backend machines for map file and site point generation, before being sent onto the actual parallel implementation running on the Bluegene/l.

Work is currently ongoing to validate the UFSRAT based techniques VolRAT and UFSRGraph in the laboratory. An interesting use of UFSRAT would be to use the similarity and discriminatory abilities of the algorithm to group virtual screening results from LIDAEUS. The grouping could be used to ensure that researchers get clusters of results that are not too similar. This would stop the top hits being dominated by closely similar compounds.

Overall, it is an exciting time for computational chemists and structural biologists, in every aspect of research: the availability and abundance of computing power is having a positive impact on science. As we near the end of Moore's law (Section 2.3.1 The argument for parallelism), the computing industry is moving towards parallelism to provide people with the increase in power they have come to expect. This is seen even on the desktop of standard office grade computers, where dual core processors are now the norm, computers built for gaming making regular use of quad core configurations. The move to parallel computing does however represent a challenge that we are yet to see if we can master. Software will not simply run faster when provided with more processors unless it is developed in such a way as to make

use of these new resources. This is the new challenge, replacing the old habit of thinking sequentially for designing, developing and moving to a more parallel mindset. The hardware technology for parallel processing is readily available; the promise of reduced runtimes and the ability to tackle larger problems rests heavily on the ability to change architectural and programming habits. Whilst some problems or at least approaches to solving problems (algorithms) appear to be unparallelisable, being too closely knit for breaking apart and forming even minuscule work units, the design of new algorithms suitable for parallelism becomes increasingly important. In the future, we may see the emergence of a class of problems with a unique property: indeed, although they do not require much computation to be solved by the future "present standards", they take an incredibly large amount of time due to their difficult nature (For example must be solved in a linear fashion – Section 2.3.2 Amdahl's Law) . With the framework for parallel computing firmly in place, the success and previously unwavering ability to solve more and more complex problems lies squarely on the shoulders of mathematicians, designers and programmers to make use of the parallel hardware available.

7 References

1. Babbage, C. (1864). *Passages from the Life of a Philosopher*, Longman, Green, Longman, Roberts, & Green.
2. Waszkowycz, B., Perkins, T. D. J., Sykes, R. A. & Li, J. (2001). Large-scale virtual screening for discovering leads in the postgenomic era. *IBM Systems Journal* **40**, 360-378.
3. Walters, W. P., Stahl, M. T. & Murcko, M. A. (1998). Virtual screening-an overview. *Drug Discovery Today* **3**, 160-178.
4. Raha, K. & Merz, K. M., Jr. (2005). Large-scale validation of a quantum mechanics based scoring function: predicting the binding affinity and the binding mode of a diverse set of protein-ligand complexes. *Journal of Medicinal Chemistry* **48**, 4558-4575.
5. Xiang, Y., Zhang, d. W. & Zhang, J. Z. (2004). Fully quantum mechanical energy optimization for protein-ligand structure. *Journal of Computational Chemistry* **25**, 1431-1437.
6. Zhang, d. W., Xiang, Y., Gao, A. M. & Zhang, J. Z. (2004). Quantum mechanical map for protein-ligand binding with application to beta-trypsin/benzamidine complex. *Journal of Chemical Physics* **120**, 1145-1148.
7. Chan, T. F. & Zheng, X. F. (2002). De novo chemical ligand design. *Drug Discovery Today* **7**, 802-803.
8. Todorov, N. P., Buenemann, C. L. & Alberts, I. L. (2006). De novo ligand design to an ensemble of protein structures. *Proteins* **64**, 43-59.
9. Buolamwini, J. K. & Assefa, H. (2002). CoMFA and CoMSIA 3D QSAR and Docking Studies on Conformationally-Restrained Cinnamoyl HIV-1 Integrase Inhibitors: Exploration of a Binding Mode at the Active Site. *Journal of Medicinal Chemistry* **45**, 841-852.
10. Cramer Iii, R. D., Patterson, D. E. & Bunce, J. D. (1988). Comparative molecular field analysis (CoMFA). 1. Effect of shape on binding of steroids to carrier proteins. *Journal of the American Chemical Society* **110**, 5959-5967.
11. Gohlke, H. & Klebe, G. (2002). DrugScore Meets CoMFA: Adaptation of Fields for Molecular Comparison (AFMoC) or How to Tailor Knowledge-Based Pair-Potentials to a Particular Protein. *Journal of Medicinal Chemistry* **45**, 4153-4170.
12. Gantchev, T. G., Ali, H. & van Lier, J. E. (1994). Quantitative Structure-Activity Relationships/Comparative Molecular Field Analysis (QSAR/CoMFA) for Receptor-Binding Properties of Halogenated Estradiol Derivatives. *Journal of Medicinal Chemistry* **37**, 4164-4176.
13. Schneider, G. & Böhm, H. J. (2002). Virtual screening and fast automated docking methods. *Drug Discovery Today* **7**, 64-70.

14. Schneider, G. & Fechner, U. (2005). Computer-based de novo design of drug-like molecules. *Nature Reviews Drug Discovery* **4**, 649.

15. Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N. & Bourne, P. E. (2000). The Protein Data Bank 10.1093/nar/28.1.235. *Nucleic Acids Research* **28**, 235-242.

16. Berman, H., Henrick, K. & Nakamura, H. (2003). Announcing the worldwide Protein Data Bank. *Nature Structural Biology* **10**, 980-980.

17. Berman, H., Henrick, K., Nakamura, H. & Markley, J. L. (2007). The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data. *Nucleic Acids Research* **35**, D301.

18. Murzin, A. G., Brenner, S. E., Hubbard, T. & Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology* **247**, 536-540.

19. Laurie, A. T. R. & Jackson, R. M. (2005). Q-SiteFinder: an energy-based method for the prediction of protein-ligand binding sites. *Bioinformatics* **21**, 1908-1916.

20. Brylinski, M., Prymula, K., Jurkowski, W., Kochanczyk, M., Stawowczyk, E. & Bourne, P. E. (2007). Prediction of Functional Sites Based on the Fuzzy Oil Drop Model. *PLoS Comput Biol* **3**, e94.

21. Koczyk, G., Wyrwicz, L. S. & Rychlewski, L. (2007). LigProf: A simple tool for in silico prediction of ligand-binding sites. *Journal of Molecular Modeling* **13**, 445-455.

22. Seckl, J. R. & Walker, B. R. (2001). Minireview: 11ß-Hydroxysteroid Dehydrogenase Type 1-A Tissue-Specific Amplifier of Glucocorticoid Action 1. *Endocrinology* **142**, 1371-1376.

23. Alberts, P., Engblom, L., Edling, N., Forsgren, M., Klingström, G., Larsson, C., Rönquist-Nii, Y., Öhman, B. & Abrahmsén, L. (2002). Selective inhibition of 11ß-hydroxysteroid dehydrogenase type 1 decreases blood glucose concentrations in hyperglycaemic mice. *Diabetologia* **45**, 1528-1532.

24. Alberts, P., Nilsson, C., Selen, G., Engblom, L. O. M., Edling, N. H. M., Norling, S., Klingstrom, G., Larsson, C., Forsgren, M. & Ashkzari, M. (2003). Selective Inhibition of 11ß-Hydroxysteroid Dehydrogenase Type 1 Improves Hepatic Insulin Sensitivity in Hyperglycemic Mice Strains. *Endocrinology* **144**, 4755-4762.

25. Webster, S. P. & Pallin, T. D. (2007). 11ß-Hydroxysteroid dehydrogenase type 1 inhibitors as therapeutic agents. *Expert Opinion on Therapeutic Patents* **17**, 1407.

26. Kim, K. W., Wang, Z., Busby, J., Tsuruda, T., Chen, M., Hale, C., Castro, V. M., Svensson, S., Nybo, R. & Xiong, F. (2007). The selectivity of tyrosine 280 of human 11ß-hydroxysteroid dehydrogenase type 1 in inhibitor binding. *FEBS Letters* **581**, 995-999.

27. Kim, K. W., Wang, Z., Busby, J., Tsuruda, T., Chen, M., Hale, C., Castro, V. M., Svensson, S., Nybo, R. & Xiong, F. (2006). The role of tyrosine 177 in human 11ß-hydroxysteroid dehydrogenase type 1 in substrate and inhibitor binding: an unlikely hydrogen bond donor for the substrate. *BBA-Proteins and Proteomics* **1764**, 824-830.

28. Walker, E. A., Clark, A. M., Hewison, M., Ride, J. P. & Stewart, P. M. (2001). Functional Expression, Characterization, and Purification of the Catalytic Domain of Human 11-ß-Hydroxysteroid Dehydrogenase Type 1. *Journal of Biological Chemistry* **276**, 21343-21350.

29. Zhang, J., Osslund, T. D., Plant, M. H., Clogston, C. L., Nybo, R. E., Xiong, F., Delaney, J. M. & Jordan, S. R. (2005). Crystal Structure of Murine 11 [beta]-Hydroxysteroid Dehydrogenase 1: An Important Therapeutic Target for Diabetes. *Biochemistry* **44**, 6948-6957.

30. Zhang, Y., Zhong, X., Gjoka, Z., Li, Y., Stochaj, W., Stahl, M., Kriz, R., Tobin, J. F., Erbe, D. & Suri, V. (2008). H6PDH interacts directly with 11ß-HSD1: Implications for determining the directionality of glucocorticoid catalysis. *Archives of Biochemistry and Biophysics* **483**, 45-54.

31. Connolly, M. L. (1983). Solvent-accessible surfaces of proteins and nucleic acids. *Science* **221**, 709.

32. Connolly, M. L. (1983). Analytical molecular surface calculation. *Journal of Applied Crystallography* **16**, 548-558.

33. Yuan, C., St. Jean, D. J., Liu, Q., Cai, L., Li, A., Han, N., Moniz, G., Askew, B., Hungate, R. W. & Johansson, L. (2007). The discovery of 2-anilinothiazolones as 11ß-HSD1 inhibitors. *Bioorganic & Medicinal Chemistry Letters* **17**, 6056-6061.

34. Wilson, K. P., Yamashita, M. M., Sintchak, M. D., Rotstein, S. H., Murcko, M. A., Boger, J., Thomson, J. A., Fitzgibbon, M. J., Black, J. R. & Navia, M. A. (1995). Comparative X-ray structures of the major binding protein for the immunosuppressant FK506 (tacrolimus) in unliganded form and in complex with FK506 and rapamycin. *Biological Crystallography* **51**, 511-521.

35. McKeon, F. (1991). When worlds collide: immunosuppressants meet protein phosphatases. *Cell* **66**, 823-826.

36. Marx, S. O., Jayaraman, T., Go, L. O. & Marks, A. R. (1995). Rapamycin-FKBP Inhibits Cell Cycle Regulators of Proliferation in Vascular Smooth Muscle Cells. *Circulation Research* **76**, 412-417.

37. Hamilton, G. S., Huang, W., Connolly, M. A., Ross, D. T., Guo, H., Valentine, H. L., Suzdak, P. D. & Steiner, J. P. (1997). FKBP12-binding domain analogues of FK506 are potent, nonimmunosuppressive neurotrophic agents in vitro and promote recovery in a mouse model of parkinson's disease. *Bioorganic & Medicinal Chemistry Letters* **7**, 1785-1790.

38. Poulter, M. O., Payne, K. B. & Steiner, J. P. (2004). Neuroimmunophilins: A novel drug therapy for the reversal of neurodegenerative disease? *Neuroscience* **128**, 1-6.

39. Steiner, J. P., Hamilton, G. S., Ross, D. T., Valentine, H. L., Guo, H., Connolly, M. A., Liang, S., Ramsey, C., Li, J. H. J. & Huang, W. (1997). Neurotrophic immunophilin ligands stimulate structural and functional recovery in neurodegenerative animal models. *Proceedings of the National Academy of Sciences* **94**, 2019-2024.

40. Fulton, K. F., Jackson, S. E. & Buckle, A. M. (2003). Energetic and structural analysis of the role of tryptophan 59 in FKBP12. *Biochemistry* **42**, 2364-2372.

41. Stebbins, J. L., Zhang, Z., Chen, J., Wu, B., Emdadi, A., Williams, M. E., Cashman, J. & Pellecchia, M. (2007). Nuclear magnetic resonance fragment-based identification of novel FKBP12 inhibitors. *Journal of medicinal chemistry* **50**, 6607.

42. Blackburn, E. (2009). Biophysical Studies of Protein-ligand Interactions and the Discovery of FKBP12 Inhibitors, Thesis authored by Liz Blackburn Submitted 2009, University of Edinburgh.

43. DeLano, W. L. (2002). The PyMOL Molecular Graphics System. DeLano Scientific, San Carlos, CA.

44. ChemAxon. (2006). Marvin. ChemAxon.

45. Intel® Core™ i7 Processor Extreme Edition and Intel® Core™ i7 Processor. Datasheet, V.

46. Moore, G. E. (1998). Cramming More Components Onto Integrated Circuits. *Proceedings of the IEEE* **86**, 82-85.

47. Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics* **38**, 114-117.

48. Transcript, V. (2005). Excerpts from a Conversation with Gordon Moore: Moore' s Law. *Intel Corporation*.

49. Flynn, M. J. (1966). Very high-speed computing systems. *Proceedings of the IEEE* **54**, 1901-1909.

50. Flynn, M. J. & Rudd, K. W. (1996). Parallel Architectures. *ACM Computing Surveys* **28**, 67 - 70.

51. Duncan, R. (1990). A survey of parallel computer architectures. *Computer* **23**, 5-16.

52. http://www.hector.ac.uk/. (2008). HECToR. Visited 10/1/2010.

53. Moreira, J. E., Midkiff, S. P., Gupta, M., Center, I. & Heights, Y. (1998). A comparison of Java, C/C++, and Fortran for numerical computing. *IEEE Antennas and Propagation Magazine* **40**, 102-105.

54. Ritchie, D. M. (1993). The development of the C language. *ACM SIGPLAN Notices* **28**, 201-208.

55. Stroustrup, B. (1993). A history of C++: 1979–1991. *ACM SIGPLAN Notices* **28**, 271-297.

56. Bull, J. M., Smith, L. A., Pottage, L. & Freeman, R. (2001). Benchmarking Java against C and Fortran for scientific applications. *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, 97 - 105.

57. Agarwal, R. C., Gustavson, F. G., McComb, J. & Schmidt, S. (1989). Engineering and scientific subroutine library release 3 for IBM ES/3090 vector multiprocessors. *IBM Systems Journal* **28**, 345-350.

58. Lindahl, E., Hess, B. & van der Spoel, D. (2001). GROMACS 3.0: a package for molecular simulation and trajectory analysis. *Journal of Molecular Modeling* **7**, 306-317.

59. Van Der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A. E. & Berendsen, H. J. C. (2005). GROMACS: Fast, Flexible, and Free. *Journal of Computational Chemistry* **26**, 1701 - 1718.

60. Berendsen, H. J. C., van der Spoel, D. & van Drunen, R. (1995). GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications* **91**, 43-56.

61. Phillips, J. C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R. D., Kale, L. & Schulten, K. (2005). Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry* **26**, 1781 - 1802.

62. Stallman, R. (1999). *Using and Porting the GNU Compiler Collection*, Free Software Foundation.

63. Lawson, C. L., Hanson, R. J., Kincaid, D. R. & Krogh, F. T. (1979). Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software* **5**, 308 - 323.

64. Frigo, M. & Johnson, S. G. (1998). FFTW User's Manual. *Massachusetts Institute of Technology, September*.

65. Schildt, H. (1995). C: The Complete Reference, Osborne. McGraw-Hill, Berkeley, CA.

66. Stroustrup, B. (1986). *The C++ programming language.(Repr. with corrections)*, Addison-Wesley Series in Computer Science.

67. Seamons, K. E. & Winslett, M. (1994). An efficient abstract interface for multidimensional array I/O. *Proceedings of the 1994 ACM/IEEE conference on Supercomputing* **22**, 650 - 659.

68. Metcalf, M., Reid, J. K. & Cohen, M. (2004). *Fortran 95/2003 explained*, Oxford University Press.

69. Gropp, W., Lusk, E. & Skjellum, A. (1999). *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press.

70. Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. *Readings in computer architecture*, 79-81.

71. Gustafson, J. L. (1988). Reevaluating Amdahl's law. *Communications of the ACM* **31**, 532-533.

72. Dowd, K. (1993). *High performance computing*, O'Reilly & Associates, Inc. Sebastopol, CA, USA.

73. Hansen, P. B. (1972). Structured multiprogramming. *Communications of the ACM* **15**, 574-578.

74. Chandra, R. (2001). *Parallel Programming in OpenMP*, Morgan Kaufmann.

75. Bal, H. E. & Tanenbaum, A. S. (1988).

76. Klaiber, A. C. & Frankel, J. L. (1993). Comparing Data-Parallel and Message-Passing Paradigms. *Parallel Processing* **2**, 11-20.

77. Goldberg, D. (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys* **23**, 5 - 48.

78. (1985). IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Std 754-1985*.

79. (2008). IEEE Standard for Floating-Point Arithmetic. *IEEE Std. 754-2008*, 1-58.

80. Leach, A. R. (1996). *Molecular modelling*, Longman Harlow, England.

81. Kearsley, S. K. (1989). On the orthogonal transformation used for structural comparisons. *Acta Crystallographica Section A: Foundations of Crystallography* **45**, 208-210.

82. Clark, M., Cramer Iii, R. D. & Van Opdenbosch, N. (1989). Validation of the general purpose Tripos 5.2 force field. *Journal of Computational Chemistry* **10**, 982 - 1012.

83. Spessard, G. O. (1998). ACD Labs/LogP dB 3.5 and ChemSketch 3.5. *Journal of Chemical Information and Computer Sciences* **38**, 1250-1253.

84. Li, Z., Wan, H., Shi, Y. & Ouyang, P. (2004). Personal Experience with Four Kinds of Chemical Structure Drawing Software: Review on ChemDraw, ChemWindow, ISIS/Draw, and ChemSketch. *Journal of Chemical Information and Computer Sciences* **44**, 1886-1890.

85. Guha, R., Howard, M. T., Hutchison, G. R., Murray-Rust, P., Rzepa, H., Steinbeck, C., Wegner, J. & Willighagen, E. L. (2006). The Blue Obelisk-Interoperability in Chemical Informatics. *Journal of Chemical Information and Modeling* **46**, 991-998.

86. The Open Babel Package.

87. Dalby, A., Nourse, J. G., Hounshell, W. D., Gushurst, A. K. I., Grier, D. L., Leland, B. A. & Laufer, J. (1992). Description of several chemical structure file formats used by computer programs developed at Molecular Design Limited. *Journal of Chemical Information and Computer Sciences* **32**, 244-255.

88. Formats, C. T. F. MDL Information Systems, Incorporated, San Ramon, CA, USA, 2005.

89. Welcome to the Worldwide Protein Data Bank, Vol. 4th January 2009.

90. Weininger, D. (1988). SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences* **28**, 31-36.

91. Weininger, D., Weininger, A. & Weininger, J. L. (1989). SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Computer Sciences* **29**, 97-101.

92. Tripos. (2001). SYBYL Mol2 File Format.

93. Lipinski, C. A., Lombardo, F., Dominy, B. W. & Feeney, P. J. (1997). Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced drug discovery reviews* **46**, 3-26.

94. Bemis, G. W. & Murcko, M. A. (1996). The Properties of Known Drugs. 1. Molecular Frameworks. *Journal of Medicinal Chemistry* **39**, 2887-2893.

95. Bemis, G. W. & Murcko, M. A. (1999). Properties of Known Drugs. 2. Side Chains. *Journal of Medicinal Chemistry* **42**, 5095-5099.

96. Lyne, P. D. (2002). Structure-based virtual screening: an overview. *Drug Discovery Today* **7**, 1047-1055.

97. Mestres, J. & Knegtel, R. M. A. (2000). Similarity versus docking in 3D virtual screening. *Perspectives in Drug Discovery and Design* **20**, 191-207.

98. Mason, J. S., Good, A. C. & Martin, E. J. (2001). 3-D Pharmacophores in Drug Discovery. *Current Pharmaceutical Design* **7**, 567-597.

99. Srinivasan, J., Castellino, A., Bradley, E. K., Eksterowicz, J. E., Grootenhuis, P. D. J., Putta, S. & Stanton, R. V. (2002). Evaluation of a Novel Shape-Based Computational Filter for Lead Evolution: Application to Thrombin Inhibitors. *Journal of Medicinal Chemistry* **45**, 2494-2500.

100. Ballester, P. J. & Richards, W. G. (2007). Ultrafast shape recognition for similarity search in molecular databases. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **463**, 1307-1321.

101. Singh, J., Chuaqui, C. E., Boriack-Sjodin, P. A., Lee, W. C., Pontz, T., Corbley, M. J., Cheung, H. K., Arduini, R. M., Mead, J. N. & Newman, M. N. (2003). Successful shape-Based virtual screening: The discovery of a potent inhibitor of the type I TGFß receptor kinase (TßRI). *Bioorganic & Medicinal Chemistry Letters* **13**, 4355-4359.

102. Carr, R. A. E., Congreve, M., Murray, C. W. & Rees, D. C. (2005). Fragment-based lead discovery: leads by design. *Drug Discovery Today* **10**, 987-992.

103. Todeschini, R. & Consonni, V. (2000). Handbook of molecular descriptors. *Weinheim: Wiley-VCH*.

104. Leo, A., Hansch, C. & Elkins, D. (1971). Partition coefficients and their uses. *Chemical reviews* **71**, 525-616.

105. Chou, J. T. & Jurs, P. C. (1979). Computer-assisted computation of partition coefficients from molecular structures using fragment constants. *Journal of Chemical Information and Computer Sciences* **19**, 172-178.

106. Iwase, K., Komatsu, K., Hirono, S., Nakagawa, S. & Moriguchi, I. (1985). Estimation of hydrophobicity based on the solvent-accessible surface area of molecules. *Chemical and pharmaceutical bulletin* **33**, 2114-2121.

107. Shave, S. R., Taylor, P., Walkinshaw, M., Smith, L., Hardy, J. & Trew, A. (2008). Ligand discovery on massively parallel systems. *IBM Journal of Research and Development* **52**, 57 - 67.

108. Woo, H. J. & Roux, B. (2005). Calculation of absolute protein-ligand binding free energy from computer simulations. *Proceedings of the National Academy of Sciences* **102**, 6825-6830.

109. Krumrine, J. R., Maynard, A. T. & Lerman, C. L. (2005). Statistical tools for virtual screening. *Journal of Medicinal Chemistry* **48**, 7477-7481.

110. Taylor, P., Blackburn, E., Sheng, Y. G., Harding, S., Hsin, K. Y., Kan, D., Shave, S. & Walkinshaw, M. D. (2007). Ligand discovery and virtual screening using the program LIDAEUS. *British Journal of Pharmacology* **153**, 555 - 567.

111. Mancera, R. L., Kallblad, P. & Todorov, N. P. (2004). Ligand-protein docking using a quantum stochastic tunneling optimization method. *Journal of Computational Chemistry* **25**, 858-864.

112. Bursulaya, B. D., Totrov, M., Abagyan, R. & Brooks, C. L., III. (2003). Comparative study of several algorithms for flexible ligand docking. *Journal of Computer Aided Molecular Design* **17**, 755-763.

113. Stahl, M. & Rarey, M. (2001). Detailed analysis of scoring functions for virtual screening. *Journal of Medicinal Chemistry* **44**, 1035-1042.

114. Schames, J. R., Henchman, R. H., Siegel, J. S., Sotriffer, C. A., Ni, H. & McCammon, J. A. (2004). Discovery of a novel binding trench in HIV integrase. *Journal of Medicinal Chemistry* **47**, 1879-1881.

115. Kramer, B., Rarey, M. & Lengauer, T. (1999). Evaluation of the FLEXX incremental construction algorithm for protein-ligand docking. *Proteins* **37**, 228-241.

116. Kellenberger, E., Rodrigo, J., Muller, P. & Rognan, D. (2004). Comparative evaluation of eight docking tools for docking and virtual screening accuracy. *Proteins Structure Function and Bioinformatics* **57**, 225-242.

117. Kitchen, D. B., Decornez, H., Furr, J. R. & Bajorath, J. (2004). Docking and scoring in virtual screening for drug discovery: methods and applications. *Nature reviews drug discovery* **3**, 935-949.

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

119. Metropolis, N. & Ulam, S. (1949). The monte carlo method. *Journal of the American Statistical Association* **44**, 335-341.

120. Morris, G. M., Goodsell, D. S., Halliday, R. S., Huey, R., Hart, W. E., Belew, R. K. & Olson, A. J. (1998). Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry* **19**, 1639-1662.

121. Goodsell, D. S., Morris, G. M. & Olson, A. J. (1996). Automated docking of flexible ligands: applications of AutoDock. *Journal of Molecular Recognition* **9**, 1-5.

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

123. Liu, M. & Wang, S. (1999). MCDOCK: a Monte Carlo simulation approach to the molecular docking problem. *Journal of Computer-Aided Molecular Design* **13**, 435-451.

124. Rarey, M., Kramer, B., Lengauer, T. & Klebe, G. (1996). A fast flexible docking method using an incremental construction algorithm. *Journal of Molecular Biology* **261**, 470-489.

125. Ewing, T. J., Makino, S., Skillman, A. G. & Kuntz, I. D. (2001). DOCK 4.0: search strategies for automated molecular docking of flexible molecule databases. *Journal of Computer Aided Molecular Design* **15**, 411-428.

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

127. Wu, S. Y., McNae, I., Kontopidis, G., McClue, S. J., McInnes, C., Stewart, K. J., Wang, S., Zheleva, D. I., Marriage, H., Lane, D. P., Taylor, P., Fischer, P. M. & Walkinshaw, M. D. (2003). Discovery of a novel family of CDK inhibitors with the program LIDAEUS: structural basis for ligand-induced disordering of the activation loop. *Structure* **11**, 399-410.

128. Ferrara, P., Gohlke, H., Price, D. J., Klebe, G. & Brooks III, C. L. (2004). Assessing Scoring Functions for Protein- Ligand Interactions. *Journal of Medicinal Chemistry* **47**, 3032-3047.

129. Wang, R., Lu, Y. & Wang, S. (2003). Comparative evaluation of 11 scoring functions for molecular docking. *Journal of Medicinal Chemistry* **46**, 303.

130. Xing, L., Hodgkin, E., Liu, Q. & Sedlock, D. (2004). Evaluation and application of multiple scoring functions for a virtual screening experiment. *Journal of Computer Aided Molecular Design* **18**, 333-344.

131. Böhm, H. J. (1992). LUDI: rule-based automatic design of new substituents for enzyme inhibitor leads. *Journal of Computer-Aided Molecular Design* **6**, 593-606.

132. Böhm, H. J. (1994). The development of a simple empirical scoring function to estimate the binding constant for a protein-ligand complex of known three-

dimensional structure. *Journal of Computer-Aided Molecular Design* **8**, 243-256.

133. Eldridge, M. D., Murray, C. W., Auton, T. R., Paolini, G. V. & Mee, R. P. (1997). Empirical scoring functions: I. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes. *Journal of Computer-Aided Molecular Design* **11**, 425-445.

134. Muegge, I. & Martin, Y. C. (1999). A General and Fast Scoring Function for Protein- Ligand Interactions: A Simplified Potential Approach. *Journal of Medicinal Chemistry* **42**, 791-804.

135. Gohlke, H., Hendlich, M. & Klebe, G. (2000). Knowledge-based scoring function to predict protein-ligand interactions. *Journal of Molecular Biology* **295**, 337-356.

136. Velec, H. F. G., Gohlke, H. & Klebe, G. (2005). DrugScoreCSD-knowledge-based scoring function derived from small molecule crystal data with superior recognition rate of near-native ligand poses and better affinity prediction. *Journal of Medicinal Chemistry* **48**, 6296-6303.

137. Charifson, P. S., Corkery, J. J., Murcko, M. A. & Walters, W. P. (1999). Consensus scoring: a method for obtaining improved hit rates from docking databases of three-dimensional structures into proteins. *Journal of Medicinal Chemistry* **42**, 5100-5109.

138. Clark, R. D., Strizhev, A., Leonard, J. M., Blake, J. F. & Matthew, J. B. (2002). Consensus scoring for ligand/protein interactions. *Journal of Molecular Graphics and Modelling* **20**, 281-295.

139. Wang, R. & Wang, S. (2001). How does consensus scoring work for virtual library screening? An idealized computer experiment. *Journal of Chemical Information and Computer Sciences* **41**, 1422-1426.

140. Schames, J. R., Henchman, R. H., Siegel, J. S., Sotriffer, C. A., Ni, H. & McCammon, J. A. (2004). Discovery of a novel binding trench in HIV integrase. *Journal of Medicinal Chemistry* **47**, 1879 - 1881.

141. Tripos. SYBYL Mol2 File Format.

142. Cohen, D. (1981). On Holy Wars and a Plea for Peace. *Computer* **14**, 48-54.

143. Goodford, P. J. (1985). A computational procedure for determining energetically favorable binding sites on biologically important macromolecules. *Journal of Medicinal Chemistry* **28**, 849-857.

144. Weiner, P. K. & Kollman, P. A. (1981). AMBER: Assisted model building with energy refinement. A general program for modeling molecules and their interactions. *Journal of Computational Chemistry* **2**, 287-303.

145. Widmer, A. (1999). WITNOTP. Novartis A.G., Basel, Switzerland.

146. Chiu, G. L. T., Gupta, M. & Royyuru, A. K. (2005). Preface. *IBM Journal of Research and Development* **49**, 191-194.

147. Boyle, P., Chen, D., Christ, N., Clark, M., Cohen, S., Cristian, C., Dong, Z., Gara, A., Joó, B. & Jung, C. (2005). Overview of the QCDSP and QCDOC computers. *IBM Journal of Research and Development* **49**, 351–365.

148. Henderson, R. L. (1995). *Job scheduling strategies for parallel processing*, Springer.

149. Bayucan, A., Henderson, R. L. & Jones, J. P. (2000). Portable Batch System Administration Guide. *Veridian System*.

150. Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B. & Lumsdaine, A. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. *Lecture Notes in Computer Science*, 97-104.

151. http://www.mesa3d.org/ Mesa Home Page. Visited 10/1/2010

152. Segal, M. & Akeley, K. (1994). The Design of the OpenGL Graphics Interface. *Silicon Graphics Computer Systems*.

153. Kilgard, M. J. (2001). NVIDIA OpenGL Extension Specifications. *NVIDIA Corporation* **6**, 7.

154. Sybyl. (2004). Version 6.91. Tripos Associates: St. Louis, MO.

155. Sheridan, R. P. (2002). The Most Common Chemical Replacements in Drug-Like Compounds. *Journal of Chemical Information and Computer Sciences* **42**, 103-108.

156. Sheridan, R. P. (2003). Finding Multiactivity Substructures by Mining Databases of Drug-Like Compounds. *Journal of Chemical Information and Computer Sciences* **43**, 1037-1050.

157. Patterson, D. E., Cramer, R. D., Ferguson, A. M., Clark, R. D. & Weinberger, L. E. (1996). Neighborhood Behavior: A Useful Concept for Validation of" Molecular Diversity" Descriptors. *Journal of Medicinal Chemistry* **39**, 3049-3059.

158. Brown, R. D. & Martin, Y. C. (1996). Use of Structure-Activity Data To Compare Structure-Based Clustering Methods and Descriptors for Use in Compound Selection. *Journal of Chemical Information and Computer Sciences* **36**, 572-584.

159. Brown, R. D. & Martin, Y. C. (1997). The Information Content of 2D and 3D Structural Descriptors Relevant to Ligand-Receptor Binding. *Journal of Chemical Information and Computer Sciences* **37**, 1-9.

160. Martin, Y. C., Kofron, J. L. & Traphagen, L. M. (2002). Do Structurally Similar Molecules Have Similar Biological Activity? *Journal of Medicinal Chemistry* **45**, 4350-4358.

161. Sheridan, R. P. & Kearsley, S. K. (2002). Why do we need so many chemical similarity search methods? *Drug Discovery Today* **7**, 903-911.

162. Lewis, R. A., Pickett, S. D. & Clark, D. E. (2000). Computer-Aided Molecular Diversity Analysis and Combinatorial Library Design. *Reviews in computational chemistry* **16**, 1-52.

163. DiMasi, J. A., Hansen, R. W. & Grabowski, H. G. (2003). The price of innovation: new estimates of drug development costs. *Journal of Health Economics* **22**, 151-185.

164. Bender, A. & Glen, R. C. (2004). Molecular similarity: a key technique in molecular informatics. *Organic & Biomolecular Chemistry* **2**, 3204-3218.

165. DiMasi, J. A. (2001). Risks in new drug development: approval success rates for investigational drugs. *Clinical Pharmacology and Theraputics* **69**, 297-307.

166. van de Waterbeemd, H. & Gifford, E. (2003). ADMET in silico modelling: towards prediction paradise? *Nature reviews drug discovery* **2**, 192-204.

167. Dogra, S. K. (2008). 'Training' From QSARWorld. A Strand Life Sciences Web Resource.

168. Silverman, B. D. & Platt, D. E. (1996). Comparative Molecular Moment Analysis (CoMMA): 3D-QSAR without Molecular Superposition. *Journal of Medicinal Chemistry* **39**, 2129-2140.

169. Karelson, M., Lobanov, V. S. & Katritzky, A. R. (1996). Quantum-Chemical Descriptors in QSAR/QSPR Studies. *Chemical reviews* **96**, 1027-1044.

170. Yoshida, F. & Topliss, J. G. (2000). QSAR Model for Drug Human Oral Bioavailability. *Journal of Medicinal Chemistry* **43**, 2575-2585.

171. Hopfinger, A. J., Wang, S., Tokarski, J. S., Jin, B., Albuquerque, M., Madhav, P. J. & Duraiswami, C. (1997). Construction of 3D-QSAR Models Using the 4D-QSAR Analysis Formalism. *Journal-American Chemical Society* **119**, 10509-10524.

172. Tong, W., Perkins, R., Xing, L., Welsh, W. J. & Sheehan, D. M. (1997). QSAR Models for Binding of Estrogenic Compounds to Estrogen Receptor a and ß Subtypes. *Endocrinology* **138**, 4022-4025.

173. Yasri, A. & Hartsough, D. (2001). Toward an Optimal Procedure for Variable Selection and QSAR Model Building. *Journal of Chemical Information and Computer Sciences* **41**, 1218-1227.

174. Baringhaus, K. H. & Hessler, G. (2004). Fast similarity searching and screening hit analysis. *Drug Discovery Today: Technologies* **1**, 197-202.

175. Holliday, J. D., Hu, C. Y. & Willett, P. (2002). Grouping of Coefficients for the Calculation of Inter-Molecular Similarity and Dissimilarity using 2D Fragment Bit-Strings. *Combinatorial Chemistry & High Throughput Screening* **5**, 155-166.

176. Willett, P., Barnard, J. M. & Downs, G. M. (1998). Chemical Similarity Searching. *Journal of Chemical Information and Computer Sciences* **38**, 983-996.

177. Carroll, F. I., Gao, Y., Rahman, M. A., Abraham, P., Parham, K., Lewin, A. H., Boja, J. W. & Kuhar, M. J. (1991). Synthesis, ligand binding, QSAR, and CoMFA study of 3. beta.-(p-substituted phenyl) tropane-2. beta.-carboxylic acid methyl esters. *Journal of Medicinal Chemistry* **34**, 2719-2725.

178. Bajorath, J. (2001). Selected concepts and investigations in compound classification, molecular descriptor analysis, and virtual screening. *Journal of Chemical Information and Computer Sciences* **41**, 233-245.

179. Ballester, P. J. & Richards, W. G. (2007). Ultrafast shape recognition to search compound databases for similar molecular shapes. *Journal of Computational Chemistry* **28**, 1711-1723.

180. Taylor, P., Blackburn, E., Sheng, Y. G., Harding, S., Hsin, K. Y., Kan, D., Shave, S. & Walkinshaw, M. D. (2007). Ligand discovery and virtual screening using the program LIDAEUS Life Sciences Review Article. *British Journal of Pharmacology*.

8 Papers by the candidate

Shave, S. R., Taylor, P., Walkinshaw, M., Smith, L., Hardy, J. & Trew, A. (2008). Ligand discovery on massively parallel systems. *IBM Journal of Research and Development* **52**, 57 - 67.

Taylor, P., Blackburn, E., Sheng, Y. G., Harding, S., Hsin, K. Y., Kan, D., Shave, S. & Walkinshaw, M. D. (2008). Ligand discovery and virtual screening using the program LIDAEUS. *British Journal of Pharmacology* **153**, 555-567

Ligand discovery on massively parallel systems

S. R. Shave
P. Taylor
M. Walkinshaw
L. Smith
J. Hardy
A. Trew

Virtual screening is an approach for identifying promising leads for drugs and is used in the pharmaceutical industry. We present the parallelization of LIDAEUS (LIgand Discovery At Edinburgh UniverSity), creating a massively parallel high-throughput virtual-screening code. This program is being used to predict the binding modes involved in the docking of small ligands to proteins. Parallelization efforts have focused on achieving maximum parallel efficiency and developing a memory-efficient parallel sorting routine. Using an IBM Blue Gene/L™ supercomputer, runtimes have been reduced from 8 days on a modest seven-node cluster to 62 minutes on 1,024 processors using a standard dataset of 1.67 million small molecules and FKBP12, a protein target of interest in immunosuppressive therapies. Using more-complex datasets, the code scales upward to make use of the full processor set of 2,048. The code has been successfully used for the task of gathering data on approximately 1.67 million small molecules binding to approximately 400 high-quality crystallographically determined ligand-bound protein structures, generating data on more than 646 million protein–ligand complexes. A number of novel ligands have already been discovered and validated experimentally.

Introduction and background

With proteins playing a major role in almost every function carried out within a cell, the ability to control or "turn off" proteins by simply blocking their binding pockets creates exciting possibilities for controlling protein activity. Molecules (such as ions, drugs, or polypeptides) that bind to a protein of interest are referred to as *ligands* and may affect the operation of the protein within an organism. Finding specific and tightly binding ligands offers the ability to alter the behavior of proteins and plays an important role in modern medicine [1, 2]. This paper describes the use of simulation as the first stage for finding novel non-covalently bound ligands for known protein structures. Virtual screening (VS) [3–5] is a broad term and includes *de novo* ligand design, that is, engineering a molecule by adding and removing groups to alter its binding properties. However, for the purposes of this paper, *VS* refers to the practice of molecular docking in which a database of ligands is used for studying potential interactions with a target receptor such as a protein-binding site. This is a commonly used approach in the first stage of the drug discovery process.

Typically, the VS process involves representing the binding pocket of a protein and then executing two steps. First, the ligand is positioned (referred to as the process of posing), and second, the resultant complex is scored. These steps are often merged or used to direct each other. Potential ligands are extracted from a database of available compounds. Software filters can then be imposed on the potential ligands to ensure that they meet certain criteria such as Lipinski's rule of five [6] or that they are not too similar [7].

Many different codes and approaches exist for simulating the docking of a ligand to a receptor, and these approaches vary greatly in their computational complexity. As mentioned, two main aspects to docking exist [8]: the positioning and global search in which the ligand is brought into a potential binding position and the scoring in which a prediction is made about the strength of present binding interactions [9, 10].

Two approaches exist for ligand positioning: flexible body docking and rigid-body docking. Flexible body docking typically treats the ligand as flexible and the protein as rigid. Sometimes, however, both the ligand and

©**Copyright** 2008 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied by any means or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8646/08/$5.00 © 2008 IBM

a portion of the protein are considered flexible structures. Flexible docking studies can be useful, simulating on the order of hundreds to thousands of protein–ligand complexes. The least computationally expensive of the approaches is rigid-body docking. Here, both ligands and protein receptors are rigid, and docking occurs through translation and rotation of the ligand. Many thousands of ligand–protein complexes can be simulated in this way. Combined approaches also exist and have been useful in the discovery of a novel binding trench present in the HIV (human immunodeficiency virus) integrase protein [11]. Flexible docking and rigid docking are vastly different in terms of complexity. For example, the process of exploring conformers of relatively simple molecules with three or four rotatable bonds typically requires more than 200 conformations to be sampled in order to study a full range of conformational space, even when using a broad step size. The complexity involved in treating ligands as flexible structures is a combinatorial problem that increases dramatically as conformational space is sampled more exhaustively [12]. Many different codes and approaches exist that attempt to deal with this complexity. The most widely used VS tools are GOLD (Genetic Optimization for Ligand Docking) [13], FlexX [14], DOCK [15], AutoDock [16], Glide [17, 18], and ICM (Internal Coordinate Mechanics) [19]. All of these codes carry out flexible ligand docking, using a range of approaches such as genetic algorithms, incremental construction, simulated annealing, and Monte Carlo methods in order to address the issue of computational complexity. Overviews of methods that address flexibility, along with limited comparisons between methods, are available [20, 21].

With most of the above program codes, a choice of scoring functions is available. Scoring functions are used in an attempt to estimate the binding energy between protein and ligand. Many different implementations exist, but each can be placed in one of three categories: force field-based methods, empirical scoring functions, and statistical knowledge-based methods [22]. A number of studies have shown that none of these methods properly predicts, or accurately estimates, the true binding energy, although each method takes a different approach for analyzing the same problem. Two studies compared seven scoring functions [8, 23], and another studied eleven [24]. For example, the scoring function of FlexX highly scores hydrogen-bonded ligands and tends to neglect lipophilic binding effects [9]. Because the differences between scoring functions are so great, results from one screening run are not directly comparable to those from another. However, a ligand may have its relative ranking compared [25]. The diversity exhibited by scoring functions has been used in consensus scoring. Using drastically different but well-performing scoring functions, the accuracy of consensus methods can be greater than that of their component parts [23, 26–28]. However, the risks associated with consensus methods are well known. "Artificial enrichment" is the main danger; that is, the scoring functions are sometimes chosen to perform well on a specific protein–ligand complex [29]. Also, various contributing functions must be well understood, and a balance must be achieved so that positive aspects of one algorithm are not diminished by another algorithm [30].

With sufficient computing time, it is possible to score protein–ligand complexes accurately using approaches derived from quantum mechanics [31–33]. The work documented in this paper is concerned with what is commonly known as high-throughput VS (HTVS), using simplified methods (i.e., methods that use neither quantum mechanical nor complex molecular dynamics approaches) to achieve higher simulation throughput [34] of complexes. The term HTVS includes program codes and approaches that test on the order of thousands of protein–ligand complexes in one program run; the simulation is not set up to produce results for just one complex.

An exhaustive study of well-known VS programs, using their default scoring functions, has shown that the program ICM, which uses metropolis Monte Carlo techniques, coupled with a force field to explore and score multiple conformations of a ligand, is the best performer (i.e., has the smallest root-mean-squared deviation from known crystallographic structures) when used to recreate the structure of known protein–ligand complexes [19]. However, this program code may not be applicable to high-performance approaches that involve the screening of large databases of small ligands.

Most program codes do not address the possibility of ligands bonding covalently to the protein receptor. The codes that do address this feature, using branches of quantum mechanics, are not high-throughput approaches. Some high-throughput codes such as FlexX can be directed to bind the ligand covalently, but this requires specific knowledge of the complex and is not performed automatically [14].

A very large increase in the number of determined protein structures (driven partly by improvements in x-ray crystallography techniques) has recently been observed. The most widely used resource for structural information on proteins is the Protein Data Bank (PDB) of the Research Collaboratory for Structural Bioinformatics (RCSB) [35]. With currently more than 40,000 protein structures in the repository, *in silico* techniques have an ever-increasing number of target structures with which to work. (Multiple entries for structures are deposited in the database as new techniques improve the resolution of those previously submitted.

Also, the proteins exist in different conformations or with different co-crystallized ligands.) Protein structure information comes in various formats, and the most widely used format is the PDB file format. This allows the representation of multiple molecules in a complex, such as a protein bound to its natural ligand.

Depending on the results obtained from the screening of thousands of potential ligands, and after further investigation, it may be worth testing the binding properties of these ligands in a biochemistry laboratory [36].

Because of the computational complexity of ligand-binding studies and the amount of high-performance computing (HPC) time typically available to biologists, only recently have VS runs been regularly breaking the one million receptor–ligand complexes "barrier" [37]. In recent years, the University of Edinburgh has been able to provide its scientists with access to teraflop computing resources. This provides researchers with a significant increase in available CPU cycles, without the complexity of applying for time with the U.K. National HPC Service. This has been achieved through the purchase of an IBM Blue Gene/L* (BG/L) supercomputer, which uses a novel architecture that is designed to provide unprecedented computing performance, coupled with very low power consumption, floor space, and cost. Designed to be highly scalable with low-latency communications, this system offers a significant breakthrough in supercomputing technology.

Although the resulting code is designed to be portable and run on any large-scale HPC platform, the target architecture for this work has been the BG/L platform. A detailed description of the architecture and its design approach have been reviewed elsewhere and will not be repeated here [38]. However, certain aspects of the architecture are important to the overall design of the parallel LIDAEUS (LIgand Discovery At Edinburgh UniverSity) code and are summarized.

The BG/L supercomputer is based on a classical massively parallel processor (MPP), distributed-memory architecture. The BG/L platform has a system-on-a-chip design, based on the PowerPC* 440 core. This feature allowed designers to have some customization capabilities without the cost of developing a totally new architecture, and it offers a customized memory system, network interface, and floating-point unit. This approach was successfully pioneered in the QCDOC (quantum chromodynamics-on-a-chip) machine, a special-purpose system for quantum chromodynamics [39]. The resulting BG/L architecture contains a large number of simple, inexpensive low-power processors connected via a custom-designed high-performance interconnect. One node (also known as a *chip*) contains two PowerPC 440 cores, with associated caches and a low clock frequency (700 MHz). The relatively low clock frequency is an important feature of the BG/L, exploiting the fact that many HPC applications are memory bound, rather than CPU bound. By using simple, inexpensive low-clock-frequency processors, the amount of power generated is low. This in turn allows for a high physical density of processors, and thus, a relatively small amount of floor space is required. This does, however, lead to two important considerations when parallelizing the LIDAEUS code. First, the code must scale to large numbers of processors or little performance benefit will be seen because of the low-clock-speed processor. Second, the code must scale well in memory, to avoid the relatively small amount of memory per processor, which limits the problem that can be studied.

The high-performance interconnect of the BG/L platform has multiple networks for different tasks. The main network is a three-dimensional (3D) torus with each node connected by six links to its nearest neighbors and utilized for point-to-point communications. These links are reasonably low latency (3.5 μs) and have a modest bandwidth (on the order of 160 MB/s) [40]. The second network is a global combining, broadcast binary tree network for collective communications (with 5-μs latency [40]). The third network is a binary tree network designed specifically for fast barrier synchronization (1.5-μs latency [40]). The two remaining networks are Ethernet networks. One is for I/O and the other is for diagnostics. The relatively low latency of the interconnect is beneficial for the LIDAEUS parallelization, helping to allow the code to scale to higher processor counts.

The hardware

The Edinburgh BG/L system is a single-rack IBM BG/L system with 1,024 chips (2,048 700-MHz PowerPC 440 processors). Installed in December 2004, this was the first BG/L system in Europe. The machine can operate either in co-processor (CO) mode, in which only one processor in each node is utilized for computation and the other is reserved for communication, or in virtual node (VN) mode, in which all processors are able to act as virtual nodes and address both computation and communication. In VN mode, the resources of the node must be shared between the two processors. Each core has a 32-KB-data, 32-KB-instruction L1 cache with no coherency between the two cores. Each node also has a very small (2 KB, 128-byte-line) L2 cache, a 4-MB shared L3 cache, and 512 MB of main memory. The system has one I/O node for every eight compute chips and four IBM BladeCenter* JS20 front-end systems for compilation and batch submission. The system runs driver release 3, with support for XL Fortran 10.1 and XL C/C++ 8.0.

LIDAEUS

LIDAEUS is a highly adaptable and modular rigid-body docking VS code written in C++ and parallelized using the Message Passing Interface (MPI). This ensures portability to a range of parallel platforms. LIDAEUS focuses on extremely high simulation speed, with the goal of a single processor being able to process a ligand in a few seconds. Extensive testing and cross-platform trials have ensured that the code is stable on a range of processors and platforms. It has been used successfully to identify a family of novel cyclin-dependent kinase inhibitors [41]. Results will be published soon that document six laboratory-confirmed cyclophilin-A binders that were found using information obtained from the parallelized code discussed here [42].

LIDAEUS job preparation is a serial process that is carried out on a local non-HPC Linux** system. Using a PDB file to represent the target protein, the first stage of the job preparation is to profile certain energy values present. These energies are written to map files, and the internal representation is a finely spaced cubic lattice with cells spaced typically 0.5 Å apart. This gives a volumetric representation of specific energies associated with the space surrounding the protein. Three maps describe van der Waals, hydrophobic, and hydrogen-bonding energies [41]. Points in the volumes have their energies defined in the following way. The van der Waals (vdw) map is defined as follows by the classic Lennard–Jones potential:

$$E(\text{vdw}) = \sum_{n}^{1} \left[A/r_{\text{p}}^{12} - B/r_{\text{p}}^{6} \right].$$

The sum is over set n where $1 - n$ are all atoms not forming hydrogen bonds with a probe; r_{p} is the separation between the protein atom and the probe (p); and A and B are coefficients from the Amber molecular dynamics package [43]. The hydrogen-bond-donor and hydrogen-bond-acceptor energy maps are defined in a similar way as the van der Waals map but include a weighting term dependent on the deviation from ideal hydrogen-bond angles. This weighting is encoded by high energy values in the map file, which can be visualized as a halo of ideal hydrogen-bonding angles when visualized by using isosurfaces at a suitable energy level.

The next stage of job preparation involves site point generation, that is, the generation of a collection of a special set of points located near the binding site. The map files typically represent the entire volume occupied by the protein and its surrounding area. The standard method of inhibiting a protein involves binding one or more foreign molecules to key sites known as *binding sites*. A protein may have one or more defined binding sites. Often, PDB files represent the protein complexed with its natural ligand. This natural ligand directs site point generation that defines the target pocket. The positioning of these points is determined by a search through the map files in the volume defined by the natural ligand (plus an adjustable amount of additional volume) and the energy values present. Generation of these points is flexible, but by default 170 points are found in the pocket and considered as one of three types: hydrophobic, hydrogen-bond acceptor, and hydrogen-bond donor. These types denote the type of atom that would favorably sit in a certain position (from an energy standpoint). The distribution of site points is many times denser than that of the atoms in a molecule. These points offer a way to constrain the search in ligand positioning, overlaying atoms of key types onto appropriate site points.

The action of LIDAEUS during execution can be broken down into a pipeline consisting of four modules (Preen, Pose, Score, and Sort) through which the small molecules (potential ligands) are passed. These small molecules in the SDF (structure data file) format are typically put into their vacuum minimum energy conformation and docked using this conformation. For this reason, large molecules such as peptides are unsuitable for docking using LIDAEUS. The next section of this paper briefly describes the expansion of LIDAEUS to enable flexible ligand docking. The following paragraphs discuss the main modules of LIDAEUS.

Preen—Atom hybridization state information is added to an internal representation of the molecule. *Hybridization* denotes characteristics of an atom that vary according to the bonding interactions and the atomic orbital state of those bound atoms.

Pose—This is the most computationally expensive part of the pipeline. The pose module takes its name from the action of positioning (i.e., "posing") a ligand against a target. Many poses may be generated by a potential ligand. Because LIDAEUS is a rigid-body docking code, suitable poses for the ligand are achieved through rotation and translation. Instead of iterating through all orientations and positions of the ligand at a coarse resolution, the pregenerated site points are used to constrain the positioning search. LIDAEUS generates a list of distances between bonded atoms in the ligand, as well as a list of distances between each site point (up to an adjustable cutoff value). A search tree is used in the following set of actions to explore combinations of fitting points. Each stage of depth within the tree matches ligand atoms to site points with increasing tolerance values. These values are user definable, but by default, the first four atoms are fitted with tolerances 0.02 Å, 0.04 Å, 0.06 Å, and 0.08 Å. The list of distances between bonded atoms in the ligand is checked against distances between site points. If a match is found, a test is carried out to see whether the next bonded atom in the tree fits a distance

present between nearby site points. If the depth of the tree of fitting atoms reaches a user-definable depth (by default four), the ligand in the correct position is able to satisfy the atom-type requirements described by the site points. Once it is known that a set of certain ligand atoms can overlay a set of site points, the 3D coordinates are extracted and LU factorization is used to obtain a combined rotation/translation matrix that overlays the key ligand atoms onto matching site points. This matrix is then stored with the representation of the ligand. Applying this matrix to the 3D coordinates of the ligand produces a pose. With ligands able to generate multiple poses, the computational complexity of the problem bears little relation to the number of ligands being tested.

Score—For each pose entry present in the representation of the potential ligand, the combined translation and rotation matrix is applied to the molecule. With the molecule oriented into a potential binding position, the energy maps are used to sum the energy contributions being made by each atom.

The resulting score has units of kcal mol^{-1}. A six-dimensional conjugate gradient energy minimization is then applied to the molecule as directed by the predicted binding score in an attempt to refine the previously generated pose. This process is repeated for each pose entry belonging to a molecule.

Sort—LIDAEUS maintains a record of the best-scoring poses across all potential ligands. It is important to note that the score denotes the energy change in binding; the more negative a binding score, the stronger the ligand is predicted to bind. A positive score predicts that energy must be added to the system in order to bind the molecule in the position dictated by its pose. Many different factors affect the range of scores achieved in the final sorted list, but a good-scoring ligand (therefore, predicted to be a strong binder) has a score less than about -20 kcal mol^{-1}.

A standard dataset for benchmarking purposes exists and contains 1.67 million small molecules (in the SDF format) that may be purchased from 24 catalogs obtained from suppliers. In this benchmark, the receptor target for these molecules is known as *FKBP12*, which plays an important role in immunosuppressive therapies. The structure is taken from the RCSB PDB and has the PDB identifier 1FKJ. This dataset is so large that it is unmanageable in a serial or low-processor-count parallel-processing environment. Selection of a random 30,000 molecules and subsequent simplistic predictive scaling calculations show that this dataset used with its target would take 29 days to analyze on an AMD Opteron** 246 Linux system with 2 GB of RAM. Runtime lengths such as this restrict users to the screening of small datasets in an effort to obtain the results they need in the amount of time available to them. HPC systems are an attractive platform for VS runs. The ultimate goal in the parallelization of LIDAEUS is to reduce the runtime for the standard dataset, thereby providing researchers with more freedom to experiment and set up even more complex runs with larger datasets.

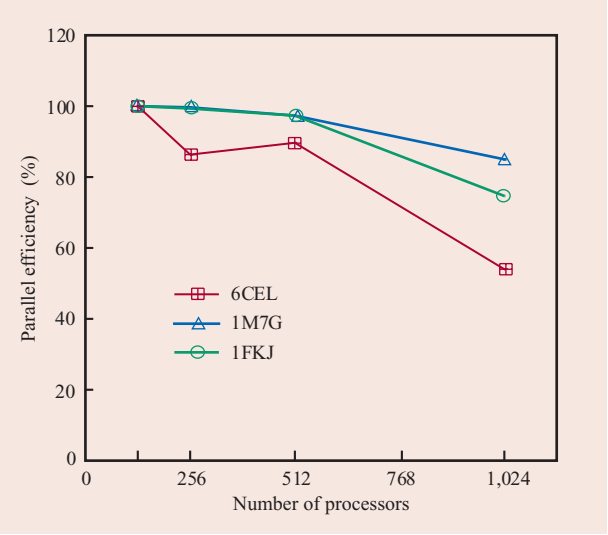
Parallelization

The original version of LIDAEUS ran on a seven-node openMosix** [44] cluster consisting of seven AMD XP 2600 processors, each with 1 GB of RAM and a standard gigabit Ethernet network acting as the interconnect. The system, therefore, consisted of a cluster of standard "desktop" machines managed by openMosix to create a cluster capable of distributed computing. This functionality is achieved by process load balancing between nodes (machines). By starting multiple processes and pipelines of LIDAEUS on each node of the cluster and running the molecular data through a simple multiplexer program, which distributed molecules to instances of the pipeline, we allowed the operating system to migrate pipelines from nodes with a high workload to nodes under a lesser load.

While this approach to parallelization enabled more results to be gathered in a shorter amount of time, the need for a truly parallel version of the code stems from the restrictiveness of the openMosix parallelization and its ability to run only in an openMosix environment. Running the code on modern HPC systems would allow the use of a large number of processors that is much greater than the seven used in the cluster.

Parallelization of LIDAEUS for the BG/L system at the University of Edinburgh takes the form of an MPI "task farm" that uses data decomposition as the parallelization strategy. Essentially, one processor runs a master process, reading in the small molecules to be complexed with the protein and then serving them to waiting worker processes residing on other processors. The worker processes run their own Preen, Pose, and Score code. As with any parallel code, computational load balancing must be addressed. Note that the code is intended for screening more than one million molecules in a single run, and the molecule data can be served to worker processes in relatively small sets ($\sim$5 to 50 molecules at a time). This data-serving process greatly limits the scope for load imbalance.

The initial parallelization was to be specifically for describing the binding pockets of a range of proteins. A selection criteria was applied to the full RCSB PDB (as of March 9, 2005), selecting crystallographically determined structures with a resolution better than 1.7 Å that made at least four hydrogen-bonding contacts with the protein and no covalent bonds between the protein and ligand. Sequence similarity searching was then used to obtain a diverse set of proteins with a sequence similarity not



for position only

Figure 1

LIDAEUS operation without sort. Parallel efficiency is studied for different numbers of processors (with 128 as the baseline) for the standard dataset using three different target receptors. The Protein Data Bank identifiers for these receptors are given in the key.

exceeding 90%. This specific problem with searching did not require sorting or retention of only the top results because information on the best-scoring minimized and non-minimized poses for each molecule was retained. The code has been used to screen a collection of 1.67 million molecules against a resultant selection of 387 protein-binding pockets. The results of this large computational run are stored in a database, and data-mining techniques are used to investigate the effectiveness of molecular descriptors in predicting binding affinities. **Figure 1** shows that the lack of a parallel sort, as well as any lack of interprocess communication, results in almost linear scaling and a parallel efficiency with up to 512 processors. With 1,024 processors, parallel efficiency for one receptor is as low as 54%. Further investigation is required, although a possible explanation for low efficiency is that for 1,023 worker processes, each writing its results to a file at unpredictable intervals (with the master process reading from disk), the network switch through which the BG/L file I/O travels becomes saturated. While the total amount of data written in a run of the standard dataset is manageable at approximately 15 GB, the constant writing of a small amount of data by 1,023 streams may explain the poor parallel efficiency. The scaling data used to produce Figure 1 suggests diminishing returns beyond 1,024 processors. A version of the code described later removes the requirement for such large amounts of I/O and subsequently shows improved scaling.

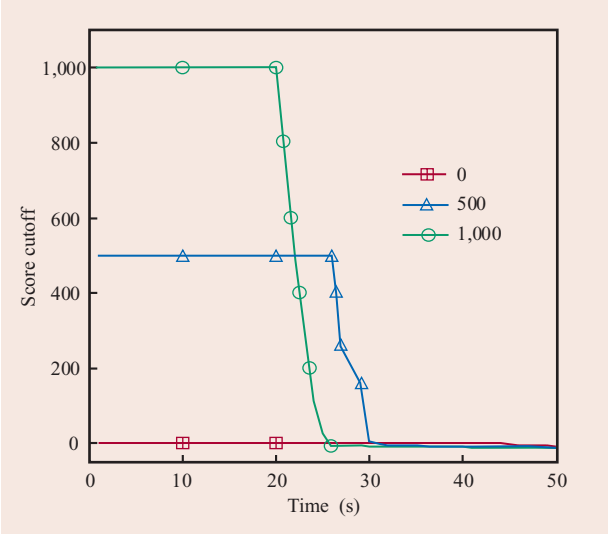
As previously stated, site points offer a way to constrain the search space in the positioning of ligands. A more exhaustive search of this space can be carried out by altering a user-specifiable variable called "resol." This variable acts as a tolerance value that denotes the distance from which the center point of key ligand atoms must be from site points. The default value of 0.02 Å may be increased to carry out a more exhaustive search. As this value increases, the computational complexity exhibits exponential growth. Massive numbers of VS runs, dealing with millions of small molecules, are not suitable for larger tolerance values. Because the goal of VS is the discovery of interesting drug leads, higher tolerance values are useful only when using VS runs on smaller targeted subsets or classes of small molecules. However, this tolerance value offers the ability to alter the amount of computation required to process a small molecule. With this in mind, along with the consideration that the limiting factor (I/O) will remain constant with an increasing number of poses, it is possible to set up runs with a larger number of poses that scale to and beyond the limit of a single BG/L cabinet utilizing 2,048 processor cores.

In order for the parallel version of LIDAEUS to be used as an HTVS tool in the rigorous sense of the term HTVS, we did not collect information on the best poses for each molecule but instead implemented a sorting algorithm to produce a reduced list of the top-scoring results from all poses made by all molecules. An interesting characteristic of the BG/L architecture, specifically the current version of its compute node kernel (CNK), is that no exceptions or errors are generated when a chip has filled its local RAM and a stack overflow event occurs. The lightweight nature of the CNK means that there is also no virtual memory. Depending on the mode in which each node is run (i.e., CO or VN mode), each chip has either 512 MB or 256 MB of local memory, respectively. With the possibility of a ligand binding to a protein in a variety of poses, one small molecule may produce many positions and scores that must be retained or at least considered for retention at the sorting stage. In a restricted-memory environment such as this, a problem arises. We are attempting to sort a dataset so large that all of it cannot be held in one memory location. The sort cannot proceed on one processor. A collaborative effort among processors must be made.

Any time that more complexity is added to a parallel code, consideration must be given to its effect on the overall execution time caused by the new code interaction with existing components. In addition to the two existing classes of processor—the master and the worker—we have a new class, the sink. Conceptually, the sink sits at the bottom of the task farm into which the workers dump their results. If the workers simply performed the

screening and then indiscriminately communicated their results to the sink, the performance of the code would decrease because of the communications overhead. Therefore, it is important to reduce the amount of communications generated. In order to achieve this objective, we introduce the idea of a cutoff value. At the start of the screening process, all worker processes screen their data, filling up a small buffer. When this buffer is full (typically when it contains ~50 poses), the contents are sent to the sink, which depending on their scores, adds them to its sorted list. When the requested sort size has been achieved by the sink (e.g., a size corresponding to 1,000 molecules), a cutoff value can be calculated. A new pose that does not meet the requirements of the cutoff can be disregarded, as it will never enter the sorted list. This cutoff value is then communicated back to the workers so that this new criteria can be applied to their local buffers.

Consideration must be given as to when this cutoff value should be communicated. One approach is to allow the worker processes to communicate their buffers to the sink and then request a new cutoff value (assuming that their data changed the cutoff). In another approach, when the worker initiates communication with the sink, a cutoff is received. The worker then removes entries from its buffer that do not meet the criteria and proceeds with the send. We chose the first pattern of suggested communication. At the start of the screening process, with the cutoff value set high to allow potentially all poses to be communicated to the sink, the code has a performance decrease. As the screening progresses, the cutoff is updated and communication to the sink is reduced. A logical approach to reducing this performance decrease would seem to be to set the cutoff to allow only very good dockings to enter the sink list. This approach may be undesirable because when a sorted list of 1,000 of the best-scoring poses is requested but fewer than 1,000 poses make it past the cutoff, the list will not contain 1,000 elements. A counterintuitive result, described later, shows that when other variables, such as the worker buffer size, are set sensibly, we may want to allow a large range of dockings to be considered in order to ensure shorter runtimes. Aside from the number of processors used for the VS run, three user-tunable variables exist that relate to the parallel sort, that is, the previously mentioned score cutoff starting value, the worker buffer size, and the number of elements the final sorted list should include. Failure to specify values results in the use of predefined defaults. Interestingly, the algorithm can be described as self-tuning. As long as the worker buffer size is not set to an unsuitably small or large value (e.g., ~1 or more than 1,000), the score cutoff value, which dictates communication, converges to a suitable value no matter how large its original starting value. As a consequence, small runs can be set up to explore the behavior of the



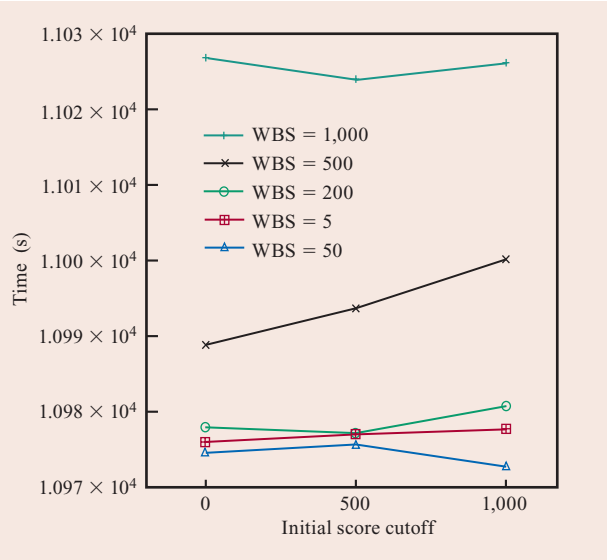
for position only

Figure 2

Using the code implementing the parallel sort, convergence of the sink score-cutoff values is shown for three different initial values of score cutoff. Data was collected using the standard dataset against the receptor represented by the Protein Data Bank identifier 1FKJ.

program when a larger proportion of its runtime is spent with an unoptimized score cutoff.

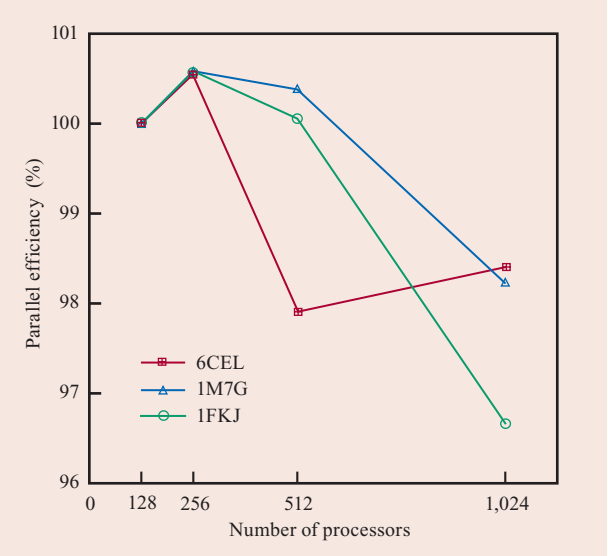
Figure 2 represents a standard scan of 1.67 million compounds against the receptor target FKBP12. When the score cutoff is started at three drastically different values (0, 500 and 1,000), convergence of the score cutoff values occurs extremely quickly. It seems counterintuitive that the larger the score cutoff value is set, the slightly shorter the overall execution time. We observed runtimes of 10,983, 10,977, and 10,973 seconds, respectively, using default settings and three different starting-score cutoff values, running on 128 nodes (256 processor cores). The speed of cutoff convergence can be seen in Figure 2, with the value for each scan at or less than 0 within 31 seconds of program execution. At this point, the lowest established cutoff is achieved by the scan using an initial value of 1,000. This observation for runs with a higher starting score cutoff is due to the fact that the contents of local buffers will be communicated many more times than those for runs with a lower cutoff. This higher rate of communication is caused by more molecules making it past the cutoff and filling the buffers earlier. As the scan with the higher starting-score cutoff progresses, the communicated buffers will typically contain a range of results varying greatly in their score. For example, some results are just able to enter the buffer through the score cutoff, while others easily enter the buffer. In another run with a more restrictive cutoff, the local buffers can be



for position only

Figure 3

The execution time is shown for the standard dataset forming a complex with a receptor (Protein Data Bank identifier 1FKJ) when 256 processors are used. A range of worker buffer sizes (WBSs) and initial score cutoffs are shown.



for position only

Figure 4

LIDAEUS operation with sort. Parallel efficiency is shown for a range of processors (128 as the baseline) for the standard dataset scanned against three different target receptors with Protein Data Bank identifiers shown in the key.

said to contain higher-quality results but are communicated less frequently. Essentially, results of poorer quality are being used to flush the good results in the local buffers through to the sink. A lack of communication keeps the sink cutoff values artificially unrestrictive. This flushing, and the ideal rate at which it should occur, is an artifact of the previously mentioned user-definable sorting variables. **Figure 3** shows runtimes for the standard dataset against a receptor with PDB identifier 1FKJ. A total of 256 processors are used for a range of worker buffer sizes and initial-score cutoffs. It is important to note that this version of the code, which implements the parallel sort, works on a different problem than the original problem with no sort. The problem for which the initial code version was used concerned the profiling of binding pockets. Here, the best-minimized and non-minimized poses for a molecule were written to disk. This is different from the action of the code when carrying out a true VS run in which the best of all poses are to be kept or sorted. Here, every pose must be considered for output to the latter stage of sorting. One molecule has the potential to generate many poses. Thus, instead of considering just two poses per molecule, multiple poses are considered. This is reflected in profiling results; the time required for processing the same dataset in both versions of the code typically is 1.5 to 1.7 times longer when each pose generated is considered for retention. Essentially, the two codes are incomparable. As can be seen from **Figure 4**, profiling the code implementing the parallel sort has resulted in almost 100% parallel efficiency up to 1,024 processors. It is worth noting that profiling shows parallel efficiency at more than 100% in some instances. This level of efficiency is due to the fact that the value is calculated using 128 processors as the baseline, and with more processors, more worker buffers are being flushed, filling the larger sink buffer and establishing a cutoff earlier. Cache effects may also influence performance.

This result supports the previous hypotheses that the code without the parallel sort is being I/O bound at higher numbers of processors. The self-optimizing nature of the sort has ensured that most of the computation, after early convergence of the score cutoff, proceeds in almost a trivially parallel manner.

Future work concerned with the LIDAEUS code will expand the program features to enable flexible ligand docking. Development will tailor the code to the lightweight massively parallel characteristics of the BG/L architecture. We aim to ensure that the increase in execution time does not become too large. We also do not want to depart from the main goal of LIDAEUS development; that is, LIDAEUS is a tool for drug lead discovery and not for predicting the exact binding mode of a ligand at the expense of increased and more

restrictive computational runtimes. While ligand flexibility is not a new concept, the novelty of LIDAEUS lies in the scale of jobs it was designed to process. Typical LIDAEUS runs will continue to simulate more than one million receptor–ligand complexes.

Conclusion

In conclusion, LIDAEUS has been parallelized and deployed on a massively parallel system, greatly reducing the amount of computational runtime required to screen compounds against a target receptor. The initial goal of the parallelization effort was to collect information on the standard dataset of 1.67 million molecules binding to 387 binding pockets. This required the simulation of more than 646 million protein–ligand complexes, as well as 387 separate LIDAEUS runs, something unobtainable without a truly parallel version of the code and access to the massive HPC resources such as those delivered by the BG/L system. The reduced runtime of LIDAEUS jobs has been key to achieving this goal.

Each version of the parallel code has reduced HTVS runtime, previously on the order of weeks to hours. The high availability of the BG/L system has made daily runs of this magnitude a reality, enabling quicker turnaround in the identification of promising leads for useful compounds.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Linus Torvalds, Advanced Micro Devices, Inc., and Amnon Barak in the United States, other countries, or both.

References

1. A. L. Hopkins and C. R. Groom, "The Druggable Genome," *Nat. Rev. Drug Discovery* **1**, No. 9, 727–730 (2002).
2. J. Drews, "Drug Discovery: A Historical Perspective," *Science* **287**, No. 5460, 1960–1964 (2000).
3. W. P. Walters, M. T. Stahl, and M. A. Murcko, "Virtual Screening—An Overview," *Drug Discovery Today* **3**, No. 4, 160–178 (1998).
4. D. B. Kitchen, H. Decornez, J. R. Furr, and J. Bajorath, "Docking and Scoring in Virtual Screening for Drug Discovery: Methods and Applications," *Nat. Rev. Drug Discovery* **3**, No. 11, 935–949 (2004).
5. P. D. Lyne, "Structure-Based Virtual Screening: an Overview," *Drug Discovery Today* **7**, No. 20, 1047–1055 (2002).
6. C. A. Lipinski, F. Lombardo, B. W. Dominy, and P. J. Feeney, "Experimental and Computational Approaches to Estimate Solubility and Permeability in Drug Discovery and Development Settings," *Advanced Drug Discovery Rev.* **46**, No. 1-3, 3–26 (2001).
7. J. R. Krumrine, A. T. Maynard, and C. L. Lerman, "Statistical Tools for Virtual Screening," *J. Medicin. Chem.* **48**, No. 23, 7477 (2005).
8. C. Bissantz, G. Folkers, and D. Rognan, "Protein-Based Virtual Screening of Chemical Databases. 1. Evaluation of Different Docking/Scoring Combinations," *J. Medicin. Chem.* **43**, No. 25, 4759–4767 (2000).
9. M. Stahl and M. Rarey, "Detailed Analysis of Scoring Functions for Virtual Screening," *J. Medicin. Chem.* **44**, No. 7, 1035–1042 (2001).
10. M. Vieth, J. D. Hirst, A. Kolinski, and C. L. Brooks, "Assessing Energy Functions for Flexible Docking," *J. Comput. Chem.* **19**, No. 14, 1612–1622 (1998).
11. J. R. Schames, R. H. Henchman, J. S. Siegel, C. A. Sotriffer, H. Ni, and J. A. McCammon, "Discovery of a Novel Binding Trench in HIV Integrase," *J. Medicin. Chem.* **47**, No. 8, 1879–1881 (2004).
12. O. Guner, O. Clement, and Y. Kurogi, "Pharmacophore Modeling and Three Dimensional Database Searching for Drug Design Using Catalyst: Recent Advances," *Current Medicin. Chem.* **11**, No. 22, 2991–3005 (2004).
13. G. Jones, P. Willett, R. C. Glen, A. R. Leach, and R. Taylor, "Development and Validation of a Genetic Algorithm for Flexible Docking," *J. Mol. Biol.* **267**, No. 3, 727–748 (1997).
14. B. Kramer, M. Rarey, and T. Lengauer, "Evaluation of the FLEXX Incremental Construction Algorithm for Protein–Ligand Docking," *Proteins: Structure, Function, and Bioinformatics* **37**, No. 2, 228–241 (1999).
15. T. J. A. Ewing, S. Makino, A. G. Skillman, and I. D. Kuntz, "DOCK 4.0: Search Strategies for Automated Molecular Docking of Flexible Molecule Databases," *Computer Aided Mol. Design* **15**, No. 5, 411–428 (2001).
16. D. S. Goodsell, G. M. Morris, and A. J. Olson, "Automated Docking of Flexible Ligands: Applications of AutoDock," *Computer Aided Mol. Design* **9**, No. 1, 1–5. (1996)
17. R. A. Friesner, J. L. Banks, R. B. Murphy, T. A. Halgren, J. J. Klicic, D. T. Mainz, M. P. Repasky, E. H. Knoll, M. Shelley, and J. K. Perry, "Glide: A New Approach For Rapid, Accurate Docking And Scoring. 1. Method and Assessment of Docking Accuracy," *J. Medicin. Chem.* **47**, No. 7, 1739–1749 (2004).
18. T. A. Halgren, R. B. Murphy, R. A. Friesner, H. S. Beard, L. L. Frye, W. T. Pollard, and J. L. Banks, "Glide: A New Approach for Rapid, Accurate Docking and Scoring. 2. Enrichment Factors in Database Screening," *J. Medicin. Chem.* **47**, No. 7, 1750–1759 (2004).
19. R. Abagyan, M. Totrov, and D. Kuznetsov, "ICM—A New Method for Protein Modeling and Design: Applications to Docking and Structure Prediction from the Distorted Native Conformation," *J. Comput. Chem.* **15**, No. 5, 488–506 (1994).
20. M. Vieth, J. D. Hirst, B. N. Dominy, H. Daigler, and C. L. Brooks, "Assessing Search Strategies for Flexible Docking," *J. Computat. Chem.* **16**, No. 14, 1623–1631 (1998).
21. R. Rosenfeld, S. Vajda, and C. DeLisi, "Flexible Docking and Design," *Biophysics Biomol. Structure* **24**, No. 1, 677–700 (1995).
22. I. Muegge and Y. C. Martin, "A General and Fast Scoring Function for Protein–Ligand Interactions: A Simplified Potential Approach," *J. Medicin. Chem.* **42**, No. 5, 791–804 (1999).
23. M. Jacobsson, P. Liden, E. Stjernschantz, H. Bostrom, and U. Norinder, "Improving Structure-Based Virtual Screening by Multivariate Analysis of Scoring Data," *J. Medicin. Chem.* **46**, No. 26, 5781–5789 (2003).
24. R. Wang, Y. Lu, and S. Wang, "Comparative Evaluation of 11 Scoring Functions for Molecular Docking," *J. Medicin. Chem.* **46**, No. 12, 2287–2303 (2003).
25. J. W. Raymond, M. Jalaie, and M. P. Bradley, "Conditional Probability: A New Fusion Method for Merging Disparate Virtual Screening Results," *J. Chem. Information Comput. Sci.* **44**, No. 2, 601–609 (2004).
26. M. Feher, "Consensus Scoring for Protein–Ligand Interactions," *Drug Disc. Today* **11**, No. 9-10, 421–428 (2006).
27. J. M. Yang, Y. F. Chen, T. W. Shen, B. S. Kristal, and D. F. Hsu, "Consensus Scoring Criteria for Improving Enrichment in Virtual Screening," *J. Chem. Information Modeling* **45**, No. 4, 1134–1146 (2005).
28. M. A. Miteva, W. H. Lee, M. O. Montes, and B. O. Villoutreix, "Fast Structure-Based Virtual Ligand Screening

Combining FRED, DOCK, and Surflex," *J. Medicin. Chem.* **48**, No. 19, 6012 (2005).
29. M. L. Verdonk, V. Berdini, M. J. Hartshorn, W. T. M. Mooij, C. W. Murray, R. D. Taylor, and P. Watson, "Virtual Screening Using Protein–Ligand Docking: Avoiding Artificial Enrichment," *J. Chemical Information Comput. Sci.* **44**, No. 3, 793–806 (2004).
30. L. Xing, E. Hodgkin, Q. Liu, and D. Sedlock, "Evaluation and Application of Multiple Scoring Functions for a Virtual Screening Experiment," *J. Computer Aided Mol. Design* **18**, No. 5, 333–344 (2004).
31. D. W. Zhang, Y. Xiang, A. M. Gao, and J. Z. H. Zhang, "Quantum Mechanical Map for Protein–Ligand Binding with Application to β-Trypsin/Benzamidine Complex," *J. Chem. Physics* **120**, No. 3, 1145–1148 (2004).
32. T. Xiang, F. Liu, and D. M. Grant, "Stochastic Dynamics of n-Nonane and Related Molecules in Solution Compared with Nuclear Magnetic Resonance Coupled Relaxation Times," *J. Chem. Physics* **95**, No. 10, 7576–7590 (2005).
33. K. Raha and K. M. Merz, Jr., "Large-Scale Validation of a Quantum Mechanics Based Scoring Function: Predicting the Binding Affinity and the Binding Mode of a Diverse Set of Protein–Ligand Complexes," *J. Medicin. Chem.* **48**, No. 14, 4558–4575 (2005).
34. H. J. Woo and B. Roux, "Calculation of Absolute Protein–Ligand Binding Free Energy from Computer Simulations," *Proc. Natl. Acad. Sci.* **102**, No. 19, 6825–6830 (2005).
35. H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The Protein Data Bank," *Nucleic Acids Res.* **28**, No. 1, 235–242 (2000).
36. J. Bajorath, "Integration of Virtual and High-Throughput Screening," *Nat. Rev. Drug Disc.* **1**, No. 11, 882–894 (2002).
37. B. Waszkowycz, T. Perkins, R. Sykes, and J. Li, "Large-Scale Virtual Screening for Discovering Leads in the Postgenomic Era," *IBM Syst. J.* **40**, No. 2, 360–378 (2001).
38. G. L. T. Chiu, M. Gupta, and A. K. Royyuru, "Preface," *IBM J. Res. & Dev.* **49**, No. 2/3, 191–194 (2005).
39. P. Boyle, D. Chen, N. Christ, M. Clark, S. Cohen, C. Cristian, Z. Dong, A. Gara, B. Joó, and C. Jung, "Overview of the QCDSP and QCDOC Computers," *IBM J. Res. & Dev.* **49**, No. 2/3, 351–365 (2005).
40. J. M. Bull, A. Gray, J. Hein, and L. Smith, "Application Performance of the Blue Gene Architecture," *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (tutorial), Tampa, FL, 2006.
41. S. Wu, I. McNae, G. Kontopidis, S. McClue, C. McInnes, K. Stewart, S. Wang, et al., "Discovery of a Novel Family of CDK Inhibitors with the Program LIDAEUS: Structural Basis for Ligand-Induced Disordering of the Activation Loop," *Structure* **11**, No. 12, 1537–1546 (2003).
42. P. Taylor, E. Blackburn, Y. Sheng, S. Harding, K. Hsin, K. Kan, S. Shave, and M. D. Walkinshaw, "Ligand Discovery and Virtual Screening Using the Program LIDAEUS," accepted for publication in *Br. J. Pharmacol.*
43. P. K. Weiner and P. A. Kollman, "AMBER: Assisted Model Building with Energy Refinement. A General Program for Modeling Molecules and Their Interactions," *J. Computat. Chem.* **2**, No. 3, 287–303 (1981).
44. "The *openMosix* Project," 2007; see *http://openmosix.sourceforge.net/*.

Received March 16, 2007; accepted for publication April 12, 2007

Steven R. Shave *Institute of Structural and Molecular Biology, School of Biological Sciences, University of Edinburgh, Michael Swann Building, King's Buildings, Mayfield Road, Edinburgh, EH9 3JR (s.r.shave@sms.ed.ac.uk).* After receiving an M.Sc. degree in high-performance computing from the Edinburgh Parallel Computing Centre (EPCC) at the University of Edinburgh (2005), Mr. Shave has continued his work on the parallelization and expansion of LIDAEUS and his current pursuit of a Ph.D. degree with the Institute of Structural and Molecular Biology. His interests include massively parallel systems and the interdisciplinary crossover between computing and biology, which have been traditionally considered two distinct fields.

Paul Taylor *Institute of Structural and Molecular Biology, School of Biological Sciences, University of Edinburgh, Michael Swann Building, King's Buildings, Mayfield Road, Edinburgh, EH9 3JR (p.taylor@ed.ac.uk).* Dr. Taylor received B.Sc. (1983) and Ph.D. (1987) degrees in chemistry from the University of Edinburgh, and after 1 year of postdoctoral research using x-ray crystallography to study amino-acid/alkaloid interactions, he became the computing officer at the Department of Biochemistry in Edinburgh University, where he supported the Protein Crystallography Research Group. He subsequently became an Infomatiker with the drug-design group at Sandoz Pharma A.G. in Basle, Switzerland, working on many aspects of protein structure and its interaction with small molecules, before returning to Edinburgh as a research fellow in the newly formed Structural Biochemistry Group at Edinburgh. He is currently interested in using x-ray crystallography to study interactions between proteins and small molecules and in design of software and database systems that perform *in silico* screening of large libraries of potential ligands against protein targets.

Malcolm Walkinshaw *Institute of Structural and Molecular Biology, School of Biological Sciences, University of Edinburgh, Michael Swann Building, King's Buildings, Mayfield Road, Edinburgh, EH9 3JR (m.walkinshaw@ed.ac.uk).* Professor Walkinshaw obtained B.Sc. (1973) and Ph.D. (1976) degrees from the Chemistry Department at the University of Edinburgh. After leading a structure-based drug design group in Sandoz in Switzerland for 10 years, he became the Chair of Structural Biochemistry in 1995 at the University of Edinburgh. He has published more than 200 papers on molecular recognition, protein structure, and drug discovery. His lab currently consists of 20 research fellows, Ph.D. students, and support staff who use crystallographic, biophysical, and computational approaches to study protein–ligand interactions.

Lorna Smith *Edinburgh Parallel Computing Centre (EPCC), School of Physics, University of Edinburgh, James Clerk Maxwell Building, King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ (l.smith@ed.ac.uk).* Dr. Smith graduated from the University of Strathclyde, Glasgow, with B.Sc. Hons. (1994) and Ph.D. (1997) degrees in pure chemistry. Her Ph.D. focused on predicting the morphology of detergent crystals and was sponsored by Unilever Research, Port Sunlight, Liverpool. She has been at EPCC since 1997 where she has been involved in research, training, and user support. Her research interests are mainly focused on investigating new languages and models for high-performance computing (HPC) and on scaling of computer applications. Dr. Smith is the HPC Research Manager at EPCC and is responsible for a wide range of projects in the HPC and computational grid area. She manages a team of scientists who are responsible for terascaling user applications on the Blue Gene/L system at EPCC and on the U.K. National HPC System, HPCx.

Judy Hardy *Edinburgh Parallel Computing Centre (EPCC), School of Physics, University of Edinburgh, James Clerk Maxwell Building, King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ (j.hardy@ed.ac.uk)*. Dr. Hardy has a B.Sc. degree in chemical physics and a Ph.D. degree in chemistry from the University of Bristol and an M.Sc. degree in software technology from Napier University in Edinburgh. Her Ph.D. research concerned microwave spectroscopy of molecules having asymmetric internal rotors. After completing her Ph.D., she worked for 10 years at Raychem Ltd., an international materials science company, in a variety of research and development roles. Dr. Hardy has worked at EPCC since 2001. She is Project Manager for a number of educational and e-learning projects and is actively involved in teaching at undergraduate and postgraduate levels within the School of Physics.

Arthur Trew *Edinburgh Parallel Computing Centre (EPCC), School of Physics, University of Edinburgh, James Clerk Maxwell Building, King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ (a.trew@ed.ac.uk)*. Prior to becoming a founder member of EPCC in 1990, Professor Trew was a Research Fellow in the Department of Physics and Astronomy at the University of Edinburgh, working on computational simulations of many-body problems. He became Director of EPCC in 1997, Deputy Director of the National e-Science Centre (NeSC) in 2001, and Professor of Computational Science in 2006. He is also a Director of UOE HPCx Ltd., a wholly owned subsidiary of the University of Edinburgh, which was formed to manage the £54M and £113M HPCx and HECToR projects delivering HPC services to the U.K. academic community.

British Journal of Pharmacology (2008) 153, S55–S67
© 2008 Nature Publishing Group All rights reserved 0007–1188/08 $30.00
www.brjpharmacol.org

REVIEW

Ligand discovery and virtual screening using the program LIDAEUS

P Taylor, E Blackburn, YG Sheng, S Harding, K-Y Hsin, D Kan, S Shave and MD Walkinshaw

The Centre for Translational and Chemical Biology, The University of Edinburgh, Michael Swann Building, King's Buildings, Mayfield Road, Edinburgh, UK

This paper discusses advances in docking and scoring approaches with examples from the high-throughput virtual screening program LIDAEUS. We describe the discovery of small molecule inhibitors for the immunophilin CypA, the cyclin-dependent kinase CDK2 and the cyclapolin series of potent Polo-like kinase inhibitors. These results are discussed in the context of advances in massively parallel computing and in the development of annotated databases.
British Journal of Pharmacology (2008) **153,** S55–S67; doi:10.1038/sj.bjp.0707532; published online 26 November 2007

Keywords: virtual screening; LIDAEUS; EDULISS; cyclophilin; cyclin-dependent kinase; Polo-like kinase

Abbreviations: CDK, cyclin-dependent kinase; CypA, human cyclophilin-A; CLogP, the octanol-water partition coefficient, calculated using the Biobyte program (http://biobyte.com.index.html) developed by Hansch and Leo; EDULISS, EDinburgh University LIgand Selection System; LIDAEUS, LIgand Discovery at Edinburgh UniverSity; MlogP, the octanol-water partition coefficient calculated as described by Moriguchi *et al*. (1992); RMSD, root mean square distance

Virtual screening overview: tools and approaches

Ligand discovery can be regarded as a simple matching problem: we would like to find a small molecule (ligand) with the appropriate shape and charge properties to bind effectively to a target protein of interest. High-throughput screening (HTS) provides one possible experimental route to a solution and libraries consisting of over 1 million compounds can be tested in days. Computational screening provides a complementary approach and with massively parallel processing, millions of compounds per week can be tested. Estimates of the number of potential small molecule drug-like compounds vary between 10^{18} and beyond 10^{63} (Lipinski and Hopkins, 2004). Consequently, for any specific target protein, even if the results from each assay and each docking run were totally reliable (which is not the case), it would still be impossible to test binding for every potential ligand. The commonly accepted Lipinski criteria (Lipinski *et al*., 1997) for orally active drug-like molecules set physicochemical property limits to increase the probability of good drug bioavailability. Drug-like molecules are expected to have a molecular weight (MW) $\leqslant$500 Da, $\leqslant$5 hydrogen bond donors (HBDs), $\leqslant$10 hydrogen bond acceptors (HBAs) and a CLog P $\leqslant$5 (the octanol-water partition coefficient calculated as described by Moriguchi *et al*. (1992) (MLogP) $\leqslant$4.15). More stringent criteria have been proposed for initial searches. For example, Lead likeness restricts MW to $<$350 Da and CLogP (the octanol-water partition coefficient, calculated using the Biobyte program (http://biobyte.com.index.html) developed by Hansch and Leo) to $<$3 (Teague *et al*., 1999). Even these more stringent cutoffs do little to reduce the astronomical numbers of potential ligands and exploring such a large-scale-matching problem will require imaginative computational and experimental approaches.

Protein targets

Recent reviews have attempted to estimate the number of druggable proteins in the Protein Data Bank (PDB) (Berman *et al*., 2000). Druggable proteins have structural features that facilitate binding to drug-like molecules. For proteins to progress from intrinsic druggability to becoming a target requires drug binding to modulate the biological role of the protein and to bring about therapeutic benefit (Fishman and Porter, 2005). Currently available literature identifies 1300 studied protein drug targets from humans and infective

Correspondence: Professor MD Walkinshaw, The Centre for Translational and Chemical Biology, The University of Edinburgh, Michael Swann Building, King's Buildings, Mayfield Road, Edinburgh, EH9 3JR, UK.
E-mail: m.walkinshaw@ed.ac.uk
Received 29 July 2007; revised 27 September 2007; accepted 4 October 2007; published online 26 November 2007

organisms (Hopkins and Groom, 2002; Russ and Lampel, 2005; Zheng *et al*., 2006). Estimates of the total number of druggable targets in the human genome have been made based on the number of disease genes; these give a total of up to 1500 targets out of 25 000 in the human genome (Hopkins and Groom, 2002). Bacterial and viral proteins also provide targets; published estimates of the number of targets from infective organisms are well over 1000. This suggests that there should be a pool of about 3000 drug targets in total (Zheng *et al*., 2006).

Of the 1300 currently studied targets, 44% are classified as enzymes, the most populated biochemical class. A total of 557 enzymes are current research targets and 134 have proved to be successful targets. Enzymes represent 50% of all successful targets (Zheng *et al*., 2006). A total of 280 research targets have experimentally determined structures with a specific drug-binding domain (represented by 107-folds), mainly by X-ray crystallography.

Within the PDB, there are about 250 uniquely different (that is $<10\%$ amino acid identity) well-determined structures in complex with 'peptide-ligands'. These represent a subset of protein–protein interactions where the interaction is controlled by a linear peptide on one side of the interface. Table 1 shows some examples of protein–peptide interactions. This group possibly represents the most druggable subset of protein–protein interactions. Short linear peptides are more amenable to replacement by small molecule mimetics. Modulating protein–protein interactions is particularly attractive due to the pivotal role of such interactions in cell signal transduction pathways and cell cycle progression (Fry and Vassilev, 2005; Chene, 2006).

There are a number of publicly accessible sources of protein–ligand binding affinities and web-based tools designed to aid the extraction of information from databases containing structural information on protein targets. For example, the BindingDB is a public, web-accessible database of measured binding affinities for biomolecules and contains data generated by isothermal titration calorimetry and enzyme inhibition (http://www.bindingdb.org/). The Relibase database (http://relibase.ebi.ac.uk/) is a web-based tool for the study of protein–ligand interaction. MSDmotif (http://www.ebi.ac.uk/msd-srv/msdmotif/) provides a tool for summarizing structural information on a database of over 6000 protein–ligand complexes found in the PDB.

Small molecule databases

A number of publicly available small molecule databases have been established over the last few years. The ligand.Info database (http://ligand.info) (Grotthuss *et al*., 2004) is a compilation of a number of publicly available databases providing a Meta-Database of over 1 million entries with calculated three-dimensional (3D) structures and some information about biological activity. The ZINC database (http://blaster.docking.org/zinc/) contains over 4.6 million commercially available compounds in various 2D and 3D formats (Irwin and Shoichet, 2005). Only compounds with MW $\leqslant$700 Da, and calculated LogP values between –4 and 6 are stored. Simple Lipinski filters or other discreet subsets of compounds can be selected.

An ambitious project financed by the National Institutes of Health has the goal of discovering sets of molecules that will specifically modulate the activities of the majority of gene product in the human and other organisms. Fast expanding databases are now being developed that contain results from a number of high-throughput screens, many of which use a set of over 100 000 chemically diverse molecules. These data are available at NCBI's database of small organic molecules at http://www.ncbi.nlm.nih.gov/sites/entrez?db=pcassay.

EDULISS, the EDinburgh University Ligand Selection System, is our in-house relational database that stores over 5 million available compounds, containing data from over 25 chemical catalogues. Of the 5.3 million compounds, 3.8 million are unique. 3D coordinates for each molecule are stored with over 1500 topological, geometric, physicochemical and toxicological descriptors per compound (Todeschini and Consonni, 2005). The descriptors can be used interactively to select subgroups of the database and also to provide profiling information. One approach to identify unique compounds is to compare the chemical graph of each compound with the graph of every other. This approach is extremely computationally expensive. An alternative method for identifying unique compounds in EDULISS's large collection has been developed. A small number of descriptors including a 3D-Wiener index, an electronegativity descriptor and a polarizability descriptor are used to group compounds. The resulting small groups of molecules with identical descriptors can then be compared using a

Table 1 Examples of protein-peptide interactions in the Protein Data Bank

PDB	*Protein*	*Peptide*	*Peptide sequence*	*Function*	*Reference*
1YCR	MDM2	p53	SQETFSDLWKLLPEN	Antitumour	(Vassilev *et al*., 2004)
1BXL (NMR)	Bcl-XL	Bak-BH3	GQVQRQLAIIGDDINR	Apoptosis	(Degterev *et al*., 2001)
1EBA	EPO	EPOR	GGTXSCHFGPLTWVCKPQGG	Anaemia	(Qureshi *et al*., 1999)
1EJ4, 1WKW	EiF4e	EiF4e-BP	RIIYDRKFLMECRN	Malignant transformation	(de and Graff, 2004)
1AXC	PCNA	p21	GRKRRQTSMTDFYHSKRRLIFS	Antitumour	(Gulbis *et al*., 1996)
1CKA	c-CRK	C3G	PPPALPPKKR	Oncogene	(Wu *et al*., 1995)
1GUX	Rb tumour suppressor	E7 peptide	DLYCYEQLN	Antitumour	(Lee *et al*., 1998)
1H9O	SH2	Penta -peptide	XVPML	Signal transduction, cancer	(Pauptit *et al*., 2001)
1QZ2	FKBP52	Hsp90	MEEVD	Steroid signalling pathways	(Wu *et al*., 2004)
1ELR	HOP	Hsp90	XMEEVD	Signalling pathways	(Scheufler *et al*., 2000)
1YVH	c-CBL	APS	GRARAVENQXSFY	Oncogene	(Hu and Hubbard, 2005)
1G3F (NMR)	XIAP-Bir3	Smac	AVPIAQKSE	Apoptosis	(Liu *et al*., 2000)

graph-matching program. A web-based interface for EDULISS has been developed; this provides a convenient way of extracting families of compounds with a user-defined set of properties.

Database profiling and compound selection

The EDULISS database comprises 25 different commercial and other smaller specialist compound collections. Of these, some 4.3 million fit the Lipinski 'rule of 5s' (Lipinski *et al*., 1997). A total of 3.2 million fit the Oprea lead-like criteria (Hann and Oprea, 2004). The more stringent Astex Rule of 3 is met by 230 000 compounds (Carr *et al*., 2005) (statistical profiles of some general descriptors are shown in Figure 1, descriptor ranges are shown in Table 2). A study by Oprea *et al*. (2007) investigated recent trends in the property space of leads, drugs and chemical probes. Leads are generally smaller, less complex and have lower LogP than drugs, due to the inevitable modifications involved in the medicinal chemistry optimization process.

It is desirable for a set of compounds for docking or assay to be selected considering both protein target and screening methodology. Solubility is of key importance for both bioavailability and 'screenability'. Experimentally derived aqueous solubility data are not available for the majority of compounds in the EDULISS database. Algorithms for predicting aqueous solubility from structure almost universally rely on a directly proportional relationship between LogP and solubility (Jorgensen and Duffy, 2002; Delaney, 2005). It might be appropriate to have a relaxed solubility requirement (MLogP $\leqslant$4.21) and to include relatively large compounds ($\leqslant$450 Da) with the aim of finding leads of high affinity and high specificity for the target. A greater range of molecular complexity may be explored with a higher MW cutoff (Schuffenhauer *et al*., 2006). However, the application of X-ray crystallography in lead discovery has different property requirements. The technique identifies fragments binding to significant regions of the target protein and then employs fragment growing or linking strategies to improve potency. Fragments are small molecules, 100–250 Da, with few functional groups (Rees *et al*., 2004; Carr *et al*., 2005; Hartshorn *et al*., 2005). In these techniques, virtual hit ligands are soaked into crystals. Relative protein concentrations are high, necessitating high ligand concentrations. Solubility problems can be compounded by the practice of soaking with a fragment cocktail to increase assay throughput. Virtual screening subsets designed for fragment screens

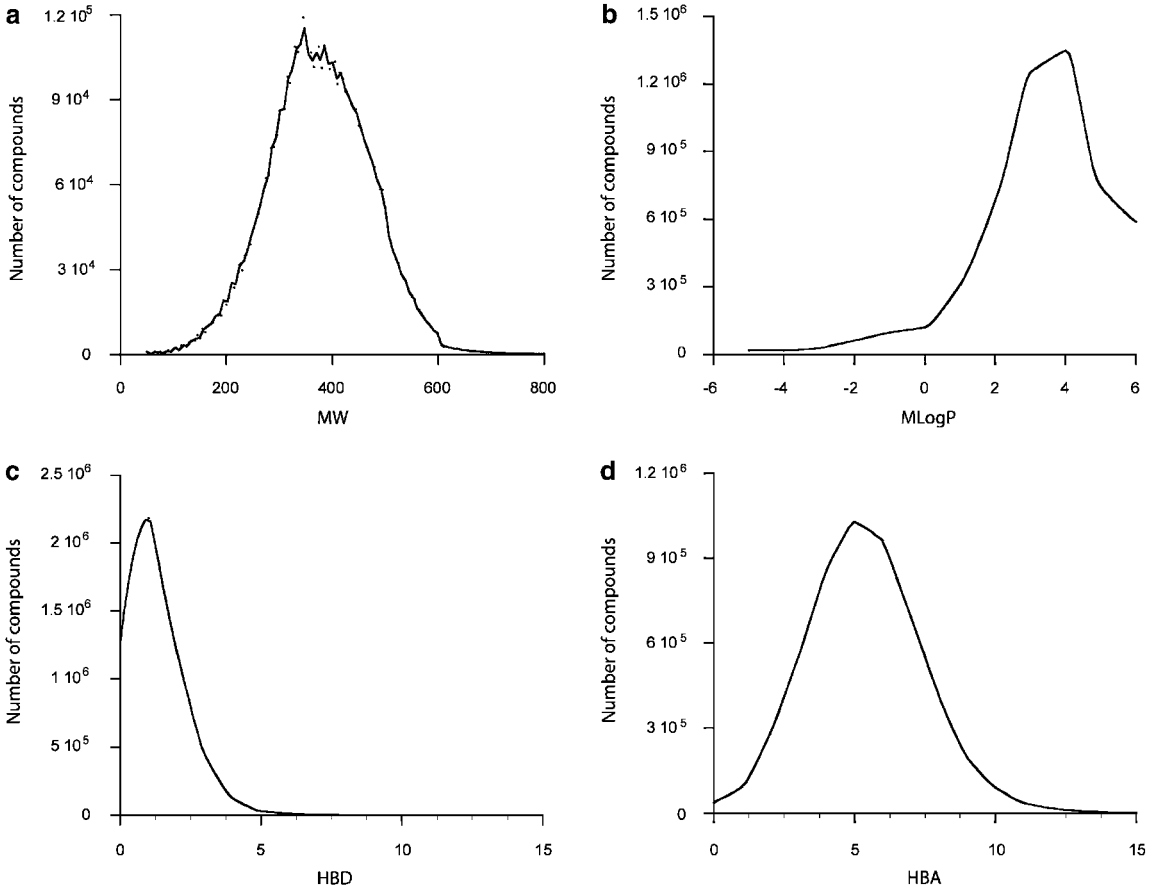


Figure 1 Molecular property profiles of 5.3 million compounds in the EDULISS database. (**a**) MW. (**b**) MLogP. (**c**) Number of HBDs. (**d**) Number of HBAs. Bin sizes for MWs are 5 Da and for MLogP, the number of HBDs and HBAs 1 U. EDULISS, EDinburgh University LIgand Selection System; HBA, hydrogen bond acceptor; HBD, hydrogen bond donor; MLogP, the octanol-water partition coefficient calculated as described by Moriguchi *et al*. (1992); MW, molecular weight

Table 2 Descriptor ranges for the EDULISS database of 5.3 million compounds

Descriptor	*Max*	*Average*	*Standard deviation*
Molecule weight	2180.59	374.28	95.77
Number of bonds	306	47.05	13.05
Number of aromatic bonds	69	13.42	5.97
Number of rings	18	3.14	1.18
Sum of atomic van der Waals volumes[a] (Å^3)	172.82	29.29	7.71
Number of rotatable bonds	77	5.31	2.73
MLogP	134.05	3.29	3.35
Topological polar surface area[b] (Å^2)	932.34	73.66	42.23

Abbreviations: EDULISS; LIgand Discovery at Edinburgh UniverSity; MLogP, the octanol-water partition coefficient calculated as described by Moriguchi (Moriguchi *et al*., 1992).
[a]Scaled on Carbon atom.
[b]Using N, O, S, P polar contributions (Todeschini and Consonni, 2005).

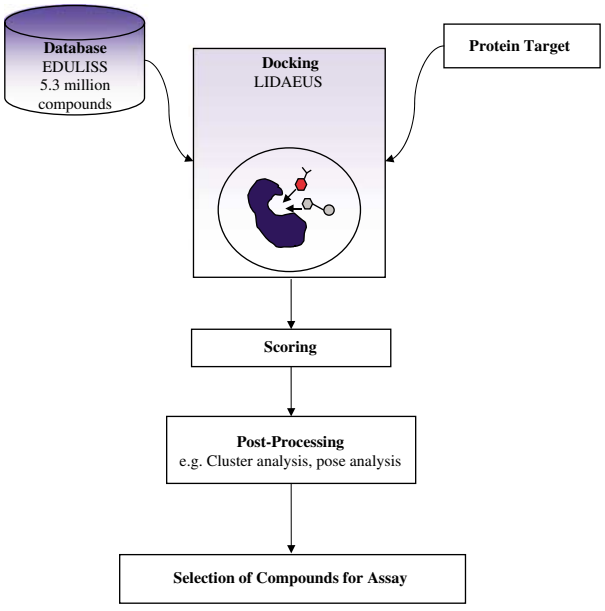


Figure 2 Steps involved in virtual screening using LIDAEUS. LIDAEUS, LIgand Discovery at Edinburgh UniverSity.

need to have stringent solubility requirements, MLogP $\leqslant 3.0$, while containing diverse scaffolds decorated with a broad range of functional groups (Moriguchi *et al*., 1992).

High-throughput virtual screening

High-throughput virtual screening achieves a high throughput of test ligands by using simplified non-quantum mechanical methods without the inclusion of complex molecular dynamics (Woo and Roux, 2005). Typically, the virtual screening process follows the steps outlined in Figure 2. A ligand is selected and positioned into the target protein-binding pocket in a given 'pose' (Muegge and Martin, 1999). The resultant complex is scored on the basis of intermolecular contacts to give a predicted strength of binding interactions (Woo and Roux, 2005). Flexible docking typically allows sampling of ligand and sometimes protein conformations during the docking procedure. Rigid body docking is however much less computationally expensive. Exploring the conformers of relatively simple molecules containing only three or four rotatable bonds (using a broad step size) requires the generation of over 200 starting conformations to be sampled in order to fully consider the majority conformational space (Guner *et al*., 2004). The most widely used flexible docking tools are GOLD (Genetic Optimization for Ligand Docking) (Jones *et al*., 1997), FlexX (Rarey *et al*., 1996; Kramer *et al*., 1999), DOCK (Ewing *et al*., 2001), AutoDock (Goodsell *et al*., 1996), Glide (Friesner *et al*., 2004; Halgren *et al*., 2004) and ICM (Internal Coordinate Mechanics) (Abagyan *et al*., 1994). A variety of different methods are used by the above tools to deal with ligand flexibility such as genetic algorithms, incremental construction, simulated annealing and Monte Carlo methods (Rosenfeld *et al*., 1995; Vieth *et al*., 1998). The diversity exhibited by scoring functions has been used in consensus scoring is implemented in, for example, X-SCORE (Wang *et al*., 2003). Using different but well-performing scoring functions, the accuracy of consensus methods can be greater than any individual scheme (Bissantz *et al*., 2000; Stahl and Rarey, 2001; Jacobsson *et al*., 2003; Raymond *et al*., 2004; Xing *et al*., 2004; Feher, 2006). However, 'artificial enrichment' is a potential pitfall, with scoring functions selected to perform well on a specific protein–ligand complex (Verdonk *et al*., 2004).

Perola *et al*. (2004) have reported that energy minimization can significantly improve the accuracy of docking poses found by GOLD (Jones *et al*., 1997) and ICM (Abagyan *et al*., 1994) programs. Our results also showed that there is better agreement between the docked pose and the crystallographic pose using rigid body refinement. A 'good fit' is defined as an root mean square distance (RMSD) $\leqslant 2$ Å between corresponding heavy atoms of the X-ray structure and the docked ligand pose.

Virtual screening has proved successful in a number of projects (Alvarez, 2004; Kitchen *et al*., 2004; Oprea and Matter, 2004; Ghosh *et al*., 2006) (Table 3). One of the major future challenges is to develop virtual screening methods capable of identifying ligands that will interrupt protein–protein interactions.

LIDAEUS as a tool for high-throughput virtual screening

LIDAEUS, LIgand Discovery at Edinburgh UniverSity, our in-house high-throughput virtual screening program (Wu *et al*., 2003) generates a grid of site points in the binding pocket of the target protein. Each site point is coloured: HBA, HBD or hydrophobic, depending on the preferred protein interaction (Figure 3).

Each molecule selected from the small molecule database is placed in the binding pocket and atoms of the docked molecule are matched to site points. An exhaustive fit of a given number of atoms from the docked molecule onto the site points is undertaken to identify reasonable poses. These are stored for subsequent rigid body energy minimization.

Table 3 Examples of hits from virtual screening experiments

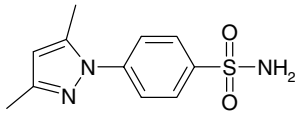
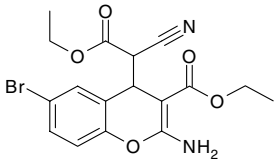
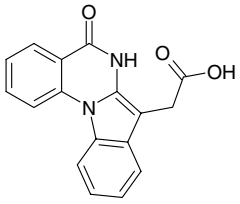
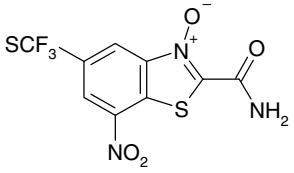
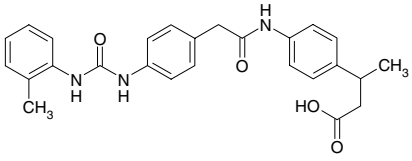
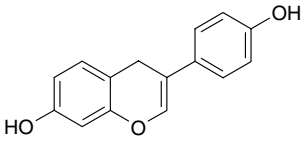
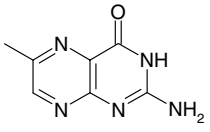
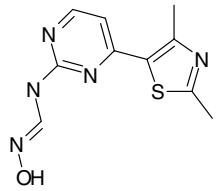
Target	*Virtual screening*	*Example structure**	*Reference*	*Assay*
Carbonic anhydrase II	FlexX (Rarey *et al.*, 1996)		(Gruneberg *et al.*, 2002)	IC_{50} Sub nanomolar
Bcl-2	DOCK (Ewing *et al.*, 2001)		(Wang *et al.*, 2000; Enyedy *et al.*, 2001)	IC_{50} 9 μM
CK2	DOCK (Ewing *et al.*, 2001)	* 	(Vangrevelinghe *et al.*, 2003)	IC_{50} 80 nM
Plk1	LIDAEUS (Wu *et al.*, 2003)		(McInnes *et al.*, 2006)	IC_{50} 20 nM
GPCR	Five targets 5-HT1A, 5-HT4, Dopamine D2, NK1, and CCR3	Compound PRX-93009 scored best for 5-HT1A, no structure shown	(Becker *et al.*, 2004)	Ki 1.0 nM
Integrin α4β1	Catalyst (Greene *et al.*, 1994)	* 	(Singh *et al.*, 2002)	IC_{50} 1.3 nM
ERβ	GOLD 2.0 (Jones *et al.*, 1997)		(Zhao and Brinton, 2005)	IC_{50} 0.68 μM
TGT	Unity/FlexX (Rarey *et al.*, 1996)		(Brenk *et al.*, 2003)	IC_{50} 0.25 μM

Table 3 *Continued*

Target	*Virtual screening*	*Example structure**	*Reference*	*Assay*
CDK2	LIDAEUS (Wu *et al*., 2003)		(Wu *et al*., 2003)	IC_{50} 2.2 μM

Structures marked with an asterisk do not represent those initially identified by *in silico* screening. Minor chemical modifications have been made and from these compounds the experimental data determined.

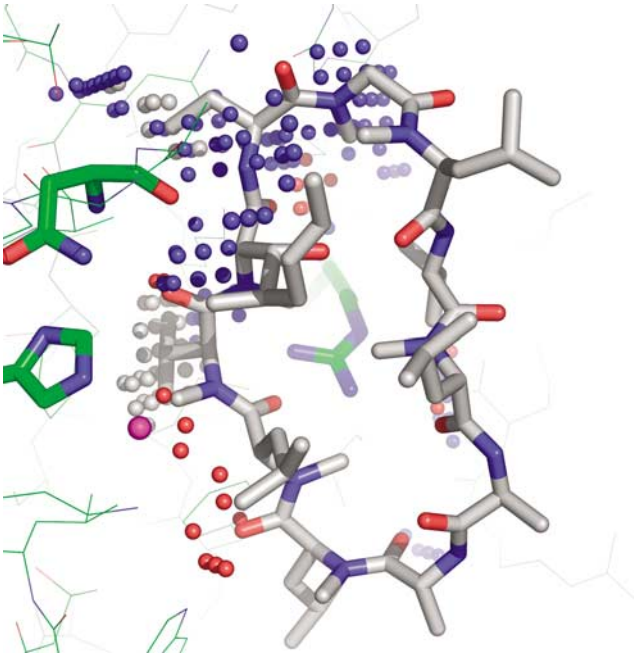


Figure 3 LIDAEUS site points in the binding pocket of CypA in complex with cyclosporine-A (PDB code 1cwa). Each site point is coloured depending on the preferred protein interaction (HBA, red; HBD, blue; hydrophobic, grey). The magenta sphere represents a key water molecule. Key residues are shown in bold. CypA, human cyclophilin-A; HBA, hydrogen bond acceptor; HBD, hydrogen bond donor; LIDAEUS, LIgand Discovery at Edinburgh UniverSity; PDB, Protein Data Bank.

There are various tunable parameters that influence how thoroughly the binding pocket is explored and hence the time required to dock a series of compounds. The precision with which an atom matches the site point, called 'resolution' in this program, is usually set between 0.02 and 0.06 Å and plays an important role in determining the number of allowed starting poses. Resolution values greater than 0.06 Å lead to an exponential growth in the number of starting poses. Increasing the number and density of the site points has a similar effect of dramatically increasing the number of allowed starting poses.

In LIDAEUS, there are two built-in scoring functions, a force field-based energy function and pose interaction profile (PIP) a knowledge-based function (Kan, 2007). The energy function is essentially a linear combination of van der Waals and hydrogen bonding energies. The geometry-dependent hydrogen bonding term incorporates salt bridges and obviates the need for calculating hydrogen atom positions. The program assigns fixed formal charges to identify ionizable groups.

The PIP score uses a protein interaction profile that can be assigned for specific regions of the binding pocket where explicit types of ligand interactions, for example, a particular hydrogen bond, are required. The PIP string is a hexadecimal code containing information about the interactions made between a given set of protein residues and the docked ligand. The target PIP string is usually based on a known X-ray crystal structure in which key features of the protein–ligand binding interaction have been identified. It is a very efficient process to match and score the bit strings of the target interactions against those calculated for the docked ligand pose. Thus, the PIP score can be used as a component of the final score to ensure that docked ligands have both favourable energies of interaction and satisfy specific interactions in their pose.

While LIDAEUS is broadly similar in function to many docking programs, it differs in two major areas: the extent to which the fitting protocol can be modified by the user and the modularity of the system. All definitions within the program in the way of atom and site point types are soft, that is, can be customized by the user. This happens at two levels: one can initially type individual atoms according to connectivity criteria and then group many or one of these types into colours used in the pose generation or scoring process. While using the default definitions allows for standard searches using atoms grouped into broad classes such as hydrophobic, HBA and HBD, it is possible to add specific restrictions.

LIDAEUS exists as a series of modules that run as a UNIX pipeline, so that initial typing of molecules, posing, scoring and sorting are all separate programs. It allows us to easily develop experimental modules and test different scoring methods. The program is being developed in two areas. The current flexible docking module is too slow to be used in a high-throughput mode and this is being addressed. Secondly, a front end is being written that allows intermediate users the ability to easily configure customizable features.

The examples discussed in the paper were run on a modest seven-node cluster. Run times are dependent on the target protein, the ligand complexity and the site point resolution set for LIDAEUS. However, representative times for a database of 50 000 ligands would be 6 h. Using an IBM Blue Gene/L supercomputer, run times have been reduced from 8 days on a seven-node cluster to 62 min on 1024 processors using a standard data set of 1.67 million small molecules.

Validation of LIDAEUS docking and scoring performance

The immunophilin proteins FKBP (FK506 Binding Protein) and human cyclophilin-A (CypA) have been used as test systems to develop and test the results from the database mining program LIDAEUS. Despite having similar biological profiles, the structure and active site of the two proteins are very different. Both proteins have peptidyl-prolyl isomerase activity and speed up the *cis–trans* equilibration of proline residues by lowering the barrier to rotation about the imide bond (Fischer *et al.*, 1993). Inhibition of the enzymatic turnover of an immunophilin substrate provides a functional assay for screening potential inhibitors (Fischer *et al.*, 1984). X-ray crystallographic, surface plasmon resonance, isothermal titration calorimetry and mass spectrometry results provide complementary techniques for characterizing ligand binding.

A set of nine chemically related ligands of human CypA with IC_{50} values between 2 and 100 μM were used to test LIDAEUS docking performance (referred to as the test set). For each ligand, the X-ray structure is known and the RMSD between corresponding atoms of the X-ray structure and the ligand structure is used as a measure of fit. Correct docking poses (RMSD $\leqslant 2$ Å) of ligand 1 in the test set (Figure 4 shows the correlation of PIP score and energy score were E, with RMSD from X-ray structure for a ligand in the test set) were all scored >0.93 by the PIP function and their energy scores, $E < -11$ kcal mol^{-1}. PIP scores are normalized between 0 and 1: a high PIP score indicates conserved interactions between those in the X-ray structure and the docked pose. The other ligands in the CypA test set have similar results, showing that a combined total score of energy function and PIP (matching a defined pose iinteraction profile) ranks the correct docking binding mode higher than alternative poses (Equation 1).

$$S = W_1 E + W_2 \sum_i PIP_i \quad (1)$$

S, total score; *E*, force field-based energy score; *PIP*, knowledge-based PIP score; W_1, weighting factor specific to protein system; W_2, weighting factor specific to protein system.

For a set of nine related cyclophilin inhibitors, the effect of changing the weights W_1 and W_2 was examined by systematically trialling different values. For this series of compounds, the values that gave the best RMSD fit of the docked pose compared to the crystallographic pose were weights of 1 and -40 for W_1 and W_2, respectively. These values proved useful in identifying a new series of cyclophilin ligands (Kan, 2007).

The role of water in accurate docking

It has been reported in several studies that water-mediated protein–ligand interactions are an important factor in the docking process. Ligands can displace water in the active site or incorporate them as an extension of the protein surface. The presence of key water molecules can significantly improve docking performance (Pospisil *et al.*, 2002; Yang and Chen, 2004). Our results show that eight out of nine ligands in the test set were correctly docked into near-native positions by LIDAEUS, while re-docking of six of them was significantly improved when key water molecules were included in the protein–ligand binding system. The presence of the key waters enables the LIDAEUS program to identify several important interactions involved in the complex and construct the significant HBA or HBD site points at the binding atom locations. (Key waters are those that form bridging H bonds to both protein and ligand molecules). Ligand 7 (Figure 5) is an example where including water improved docking performance. The presence of the key waters enables the LIDAEUS program to construct the significant HBA or HBD site points at the binding atom

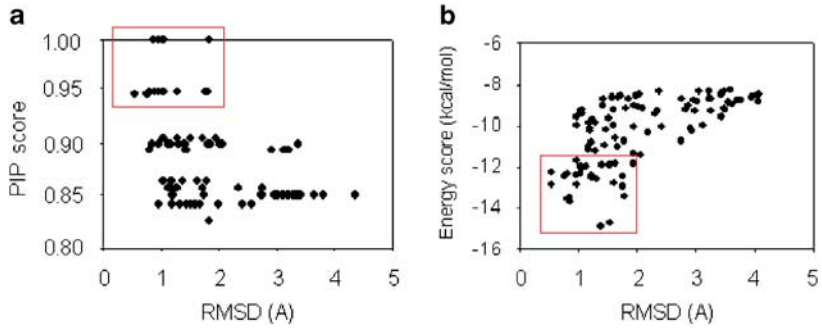


Figure 4 Correlation of PIP score and energy score, E, with RMSD from X-ray structure for ligand 1 of the CypA test set. (**a**) Plot of PIP score against RMSD of docking poses with respect to X-ray structure. (**b**) Plot of energy score, E, against RMSD with respect to X-ray structure. Red boxes highlight 'good poses' that meet scoring cutoffs: PIP score >0.93, energy score <-11 kcal mol^{-1}. CypA, human cyclophilin-A; PIP, pose interaction profile; RMSD, root mean square distance.

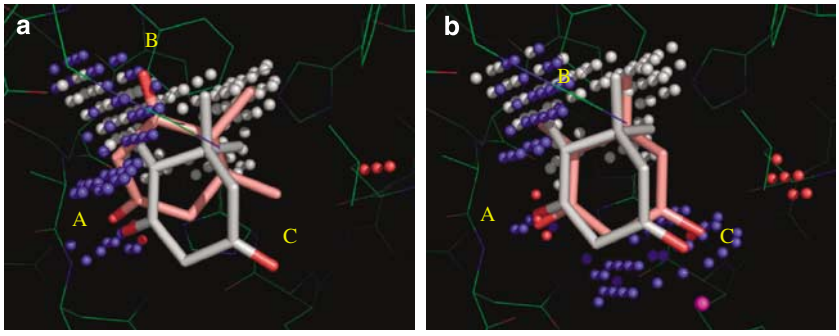


Figure 5 Including water in site point generation. Poses were improved when an essential water molecule was used in the calculation of energy maps and in site point construction. X-ray structure of the ligand is shown in white and docked poses in pink. (**a**) Illustrates how one oxygen atom (O1) from the ligand was put into the experimental position (position A), but the other oxygen atom (O2) was put into position B instead of position C, as revealed from the X-ray structure. (**b**) When the important water molecule (in magenta) is included in site point generation, competent site points set helped bring the oxygen atom to position C.

locations. Furthermore, energy maps contoured with the presence of key waters better represent the energy distribution in the local area. Thus, those maps would give the correct fits lower energy scores than maps generated without key waters.

Discovery of ligands for immunophilins

In a test to find CypA ligands, the ZINC database (Irwin and Shoichet, 2005) of 2 million compounds was used as an input for a LIDAEUS screen looking for compounds that would match site points in the active site of CypA (two parallel runs using 60 site points with a resolution of 0.06 Å and 170 site points with a resolution of 0.04 Å). The top 2000 poses were re-ranked, specifying that hydrophobic interactions with Phe113 and a hydrogen bond to Arg55 were satisfied, using PIP scoring (Figure 6).

The combined energy and PIP scores ranged from −164 to −80 (arbitrary units). The top 360 unique compounds were grouped according to chemical similarity (using molecular fingerprinting and Tanimoto coefficients) and binding mode (visually using Pymol). From this analysis, 14 compounds (all chemically distinct from known cyclophilin inhibitors) were purchased and tested for inhibition and binding by peptidyl-prolyl isomerase assay (Kofron *et al*., 1991). Eleven compounds showed a statistically significant reduction in peptidyl-prolyl isomerase activity. Six of the 14 compounds were 'hits' in the peptidyl-prolyl isomerase enzymatic assay; they inhibited CypA with IC_{50} values ranging from 27 to 135 μM. Subsequent isothermal titration calorimetry studies for the best three compounds gave K_d values of 2 to 8 μM.

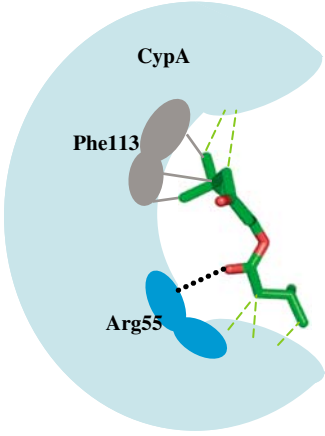


Figure 6 PIP used in the CypA virtual screening experiment. The top 2000 poses (rank ordered using the energy score, E,) were re-ranked, specifying that there were predicted hydrophobic interactions with Phe113 (grey lines) and a predicted hydrogen bond to Arg55 (black dotted lines) using PIP scoring. The diagram shows a molecule making interactions specified in the PIP. The green dashed lines are non-covalent interactions not specified in the PIP. CypA, human cyclophilin-A; PIP, pose interaction profile;

Virtual screening for CDK inhibitors using LIDAEUS

Cyclin-dependent kinases (CDKs) are key regulators in all steps of the cell cycle and as such are interesting targets for anticancer therapies. There are already a number of clinical trials underway with CDK2 and CDK4 inhibitors for a range of cancers (Collins and Garrett, 2005). The small molecule inhibitors, roscovotine (Seliciclib) and flavopiridol, are CDK2 inhibitors and show promising activity in lung cancer. These drugs target the ATP binding site of the CDKs. This is a problem in the design of CDK selective drugs, as all nine CDKs show some homology and most of the active site residues are well conserved. Another complicating factor in the design of specific inhibitors is that the active form of the kinase is induced by complex formation with a partner cyclin and phosphorylation of a specific threonine residue located on the T-loop of the kinase. These events cause subtle changes in active site geometry, which may be important for inhibitor design.

We used LIDAEUS to carry out a virtual screen of 50 000 commercially available compounds from the Maybridge catalogue (www.maybridge.com) docked into the active site of CDK2 (taken from the X-ray structure of the CDK2–staurosporine complex; PDB code 1AQ1). The predicted top 120 poses based on the docking score were screened at a fixed concentration of 30 μM using an assay to monitor the

inhibition of phosphorylation by CDK2/cyclinE. Twenty-nine percent of the compounds were classed as active by showing more than 30% inhibition. The most active four compounds all had a heteroaryl-2-amino-pyrimidine core and measured IC_{50} values between 0.9 and 17 μM (Table 4). X-ray crystal structures of the four hits were obtained and each was clearly identified in the ATP binding site. A comparison was made of the calculated docked pose (without any PIP influence) against the experimentally determined ligand structures. The four ligands were all found to dock in twisted conformations with a twist of 35° around the bond between the two aromatic rings. The RMSD atom against atom fit of the three top scoring docked ligands versus the experimental structure were 1.6, 1.58 and 3.42 Å with scores of −24, −23 and −20 kcal mol^{-1}, respectively. Despite the chemical similarity of these four ligands, they adopt different binding modes (Table 4) CYC1 and CYC2 form identical hydrogen-bond interactions to ATP: NH….O (Glu81), N….HN(Leu83) and CH….O(Leu83). When the amine group is substituted as in CYC3 and CYC4, the ATP binding mode is precluded and the ligand flips over to allow the bulky substituent to point out of the pocket. An alternative hydrogen bonding pattern is made CH…O (Glu81), N….HN(Leu83) and NH….O(Leu83). These four structures provided an excellent starting point for the design of chemical modifications. Over 40 related structures have been synthesized to optimize *in vivo* potency. The tightest binding ligand of this series, an amino derivative (CYC5), has a $K_i = 2$ nM and was shown to induce cell death in cultured HeLa cells (Wang *et al*., 2004a, b)

The importance of fine tuning a template structure in virtual screening

The shape and surface of the target pocket is clearly one of the most important factors in successful virtual screening runs. The search for CDK2-specific inhibitors highlighted the importance of understanding the biological role of the target protein. A crystallographic study was used to analyse the structures of six inhibitor ligands belonging to the thiazole-pyrimidine class, identified by LIDAEUS; both in complex with monomeric CDK2 and also with the binary CDK2/cyclinA active complex (Wu *et al*., 2003; Kontopidis *et al*., 2006). The activation of CDK2 by phosphorylation and cyclin binding causes significant loop and helix movements but leaves the shape of the ATP binding site relatively unchanged with a maximum side-chain movement between 1 and 2 Å for residues comprising this pocket. However, these small differences in pocket shape play a major role in the relative binding strengths of inhibitors. In some cases, the same ligands can adopt significantly different poses in the monomeric and active complexes. Binding enthalpies of the ligands have been estimated based on calculated van der Waals and hydrogen bond contacts measured in the crystal. The measured IC_{50} values correlate well with the calculated interaction energy (energy score) for the binary complex, but show poor correlation with the inactive complex. This fits with the way the assay has been carried out—using the active complex. It also suggests that the enthalpic energy-scoring scheme, using van der Waals and hydrogen bonding terms, provides a self-consistent measure of binding strength (Kontopidis *et al*., 2006).

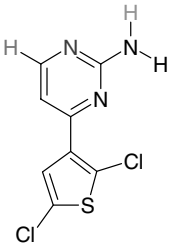
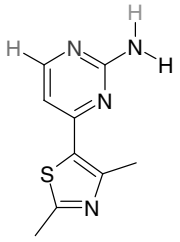
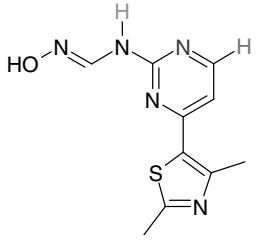
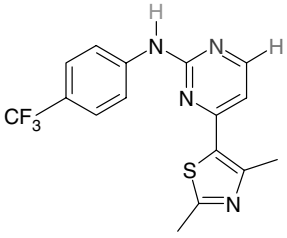
The discovery of cyclapolin, a potent Polo-like kinase inhibitor

Polo-like kinase 1 (Plk1) controls the G2/M transition of the cell cycle by phosphorylating a number of substrates that function in mitotic progression. Overexpression of Plk1 is frequently observed in tumours and in downregulation, using small interfering RNA, has been shown to inhibit cancer cell proliferation (McInnes *et al*., 2005). Small molecule Plk-specific inhibitors are valuable biological tools and can be used as leads for antitumour agents. A number of general kinase inhibitors, such as staurosporine and purvalanol, are known to inhibit Plk1 (McInnes *et al*., 2005). After years of intensive effort by academic and pharmaceutical research groups, the X-ray structure of the kinase domain has recently been published (Kothe *et al*., 2007) in complex with a pyrazole inhibitor (PHA-680626), which has an IC_{50} value of 0.5 μM (PDB code 2owb). Before this structure became available, we had developed a homology model of the kinase domain of Plk based on the staurosporine-bound conformation of protein kinase A, which has a 31% sequence identity (PDB code 1stc). The model was shown to be consistent with the known structure-activity properties of a series of ligands which were docked into the binding pocket in a similar manner to that found in CDK2. LIDAEUS was used to dock a library of 200 000 commercially available compounds into the modelled active site of Plk1. A total of 350 of the top-ranked compounds were then assayed by measuring inhibition of Plk1 phosphorylation of Cdc25C. A number of Plk1 inhibitors were identified with potencies ranging between 0.5 and 20 μM. A series of compounds (named the cyclapolins) based on the benziathole N-oxide core of the most active hit were synthesized and provide a consistent structure-activity relationships for the inhibition of Plk1 (Figure 7). The most active compound in this series showed significant improvement in potency and has an IC_{50} value of 2 nM. For this series, there is a good correlation between the docking score and potency. Treatment of HeLa cells with cyclapolin1 leads to mitotic cells that show severe spindle abnormalities (McInnes *et al*., 2006).

Outlook

The evolution of structure-based lead discovery has been guided by fashion and by some interesting technological advances. Twenty-five years ago, we had the first useful molecular graphics systems that could help medicinal chemists visualize molecular properties. This technology along with fast data collection and structure determination of protein X-ray structures opened the path to structure-based methods. Ironically, in the mid 1990s, just as this approach was beginning to bear fruit, the fashion swung to robotics and high-throughput screening, possibly spurred by the newly founded discipline of Combinatorial Chemistry,

Table 4 CDK2 inhibitors discovered using LIDAEUS

Compound	*Kinase inhibition (CDK2/ cyclin E) IC_{50} (μM)*	*Hydrogen-bonding pattern*	*Reference*
CYC1 	17	ATP hydrogen-bonding pattern: NH….O(Glu81), 2.96 Å, N….HN(Leu83), 3.64 Å, CH….O(Leu83), 3.36 Å	(Wu *et al*., 2003)
CYC2 	13	ATP hydrogen-bonding pattern: NH….O(Glu81), 2.86 Å, N….HN(Leu83), 3.30 Å, CH….O(Leu83), 3.25 Å	(Wu *et al*., 2003)
CYC3 	2.2	Alternative hydrogen bonding pattern: CH….O(Glu81), 3.31 Å, N….HN(Leu83),2.82 Å, NH….O(Leu83), 2.54 Å Ligand flips over to allow the bulky substituent to point out of the pocket	(Wu *et al*., 2003)
CYC4 	0.9	Alternative hydrogen bonding pattern: CH….O(Glu81), 3.31 Å, N….HN(Leu83), 2.92 Å, NH….O(Leu83), 2.58 Å Ligand flips over to allow the bulky substituent to point out of the pocket	(Wu *et al*., 2003)

Colour coding denotes atoms involved in key hydrogen-bonding interactions.

which made it possible to generate very large libraries of compounds. Now in larger organizations high-throughput screening, *in silico* and structure-based approaches are quite well integrated.

The main challenges in docking are still the old problems of how to efficiently model effects including dielectrics, entropy, water and flexibility. Advances in computer architectures may help tackle such problems. However, we also need to design methods that allow efficient simulation of these effects. Possibilities include using cliques of side chain conformations around the active site, and hybrid molecular modelling/quantum mechanical calculations. High-throughput virtual screening, using simplified methods (non-quantum mechanical or complex molecular dynamics), can already achieve docking rates of over 1 M compounds an hour (Shave *et al*., 2008).

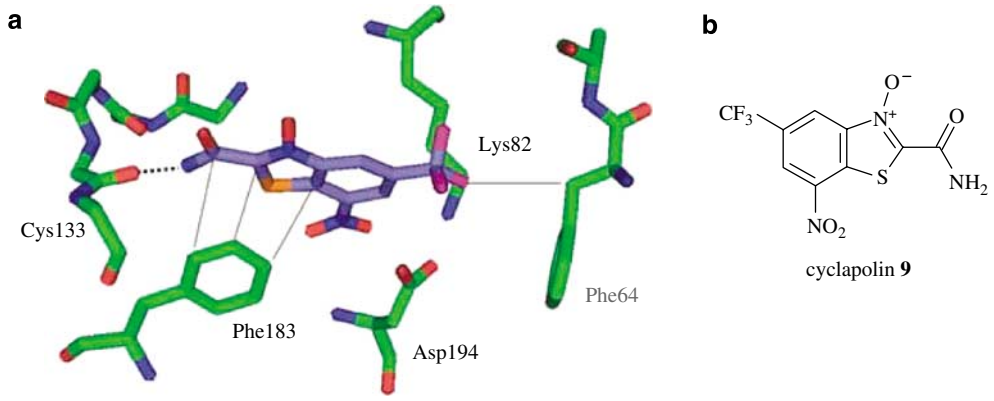


Figure 7 Cyclapolin 9 in the active site of Plk1. (**a**) Proposed binding mode of cyclapolin 9 in the ATP-binding site of Plk1, the black dotted line represents a hydrogen bonding interaction between cyclapolin and Cys133 and the grey lines hydrophobic interactions between cyclapolin and Phe183. Diagram reproduced from (McInnes *et al.*, 2006). (**b**) Cyclapolin 9 is the top hit from virtual screening, IC_{50} 500 nM. Plk1, Polo-like kinase 1.

Technical advances in miniaturization (396-well plates) and sensitive ligand-binding assays are already generating very large amounts of binding data, which contribute to structure-activity relationships. Accurate protein–ligand binding data can add to our understanding of how proteins recognize ligands. Identifying the key features of successful virtual screening calculations can only enhance the chances of discovering new ligands.

Acknowledgements

Elizabeth Blackburn acknowledges support from Organon, UK as part of a CASE studentship with the Biotechnology and Biological Sciences Research Council, UK (BBSRC). Simon Harding and Steven Shave acknowledge support from the BBSRC.

Conflict of interest

The authors state no conflict of interest.

References

Abagyan R, Totrov M, Kuznetsov D (1994). ICM—a new method for protein modeling and design: applications to docking and structure prediction from the distorted native conformation. *J Comput Chem* **15**: 488–506.

Alvarez JC (2004). High-throughput docking as a source of novel drug leads. *Curr Opin Chem Biol* **8**: 365–370.

Becker OM, Marantz Y, Shacham S, Inbal B, Heifetz A, Kalid O *et al.* (2004). G protein-coupled receptors: *in silico* drug discovery in 3D. *Proc Natl Acad Sci USA* **101**: 11304–11309.

Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H *et al.* (2000). The Protein Data Bank. *Nucleic Acids Res* **28**: 235–242.

Bissantz C, Folkers G, Rognan D (2000). Protein-based virtual screening of chemical databases. 1. Evaluation of different docking/scoring combinations. *J Med Chem* **43**: 4759–4767.

Brenk R, Naerum L, Gradler U, Gerber HD, Garcia GA, Reuter K *et al.* (2003). Virtual screening for submicromolar leads of tRNA-guanine transglycosylase based on a new unexpected binding mode detected by crystal structure analysis. *J Med Chem* **46**: 1133–1143.

Carr RA, Congreve M, Murray CW, Rees DC (2005). Fragment-based lead discovery: leads by design. *Drug Discov Today* **10**: 987–992.

Chene P (2006). Drugs targeting protein–protein interactions. *ChemMedChem* **1**: 400–411.

Collins I, Garrett MD (2005). Targetting the cell division cycle in cancer. CDK and cell cycle checkpoint kinase inhibitors. *Curr Opin Pharmacol* **4**: 366–373.

de BA, Graff JR (2004). eIF-4E expression and its role in malignancies and metastases. *Oncogene* **23**: 3189–3199.

Degterev A, Lugovskoy A, Cardone M, Mulley B, Wagner G, Mitchison T *et al.* (2001). Identification of small-molecule inhibitors of interaction between the BH3 domain and Bcl-xL. *Nat Cell Biol* **3**: 173–182.

Delaney JS (2005). Predicting aqueous solubility from structure. *Drug Discov Today* **10**: 289–295.

Enyedy IJ, Ling Y, Nacro K, Tomita Y, Wu X, Cao Y *et al.* (2001). Discovery of small-molecule inhibitors of Bcl-2 through structure-based computer screening. *J Med Chem* **44**: 4313–4324.

Ewing TJ, Makino S, Skillman AG, Kuntz ID (2001). DOCK 4.0: search strategies for automated molecular docking of flexible molecule databases. *J Comput Aided Mol Des* **15**: 411–428.

Feher M (2006). Consensus scoring for protein–ligand interactions. *Drug Discov Today* **11**: 421–428.

Fischer G, Bang H, Mech C (1984). Determination of enzymatic catalysis for the *cis–trans*-isomerization of peptide binding in proline-containing peptides]. *Biomed Biochim Acta* **43**: 1101–1111.

Fischer S, Michnick S, Karplus M (1993). A mechanism for rotamase catalysis by the FK506 binding protein (FKBP). *Biochemistry (Moscow)* **32**: 13830–13837.

Fishman MC, Porter JA (2005). Pharmaceuticals: a new grammar for drug discovery. *Nature* **437**: 491–493.

Friesner RA, Banks JL, Murphy RB, Halgren TA, Klicic JJ, Mainz DT *et al.* (2004). Glide: a new approach for rapid, accurate docking and scoring. 1. Method and assessment of docking accuracy. *J Med Chem* **47**: 1739–1749.

Fry DC, Vassilev LT (2005). Targeting protein–protein interactions for cancer therapy. *J Mol Med* **83**: 955–963.

Ghosh S, Nie A, An J, Huang Z (2006). Structure-based virtual screening of chemical libraries for drug discovery. *Curr Opin Chem Biol* **10**: 194–202.

Goodsell DS, Morris GM, Olson AJ (1996). Automated docking of flexible ligands: applications of AutoDock. *J Comput Aided Mol Des* **9**: 1–5.

Greene J, Kahn S, Savoj H, Sprague P, Teig S (1994). Chemical function queries for 3D database search. *J Chem Inf Comput Sci* **34**: 1297–1308.

Grotthuss MV, Pas J, Koczyk G, Wyrwicz LS (2004). Ligand. Info small-molecule Meta-Database. *Comb Chem High Throughput Screen* **7**: 757–761.

Gruneberg S, Stubbs MT, Klebe G (2002). Successful virtual screening for novel inhibitors of human carbonic anhydrase: strategy and experimental confirmation. *J Med Chem* **45**: 3588–3602.

Gulbis JM, Kelman Z, Hurwitz J, O'Donnell M, Kuriyan J (1996). Structure of the C-terminal region of p21(WAF1/CIP1) complexed with human PCNA. *Cell* **87**: 297–306.

Guner O, Clement O, Kurogi Y (2004). Pharmacophore modeling and three dimensional database searching for drug design using catalyst: recent advances. *Curr Med Chem* **11**: 2991–3005.

Halgren TA, Murphy RB, Friesner RA, Beard HS, Frye LL, Pollard WT *et al*. (2004). Glide: a new approach for rapid, accurate docking and scoring. 2. Enrichment factors in database screening. *J Med Chem* **47**: 1750–1759.

Hann MM, Oprea TI (2004). Pursuing the leadlikeness concept in pharmaceutical research. *Curr Opin Chem Biol* **8**: 255–263.

Hartshorn MJ, Murray CW, Cleasby A, Frederickson M, Tickle IJ, Jhoti H (2005). Fragment-based lead discovery using X-ray crystallography. *J Med Chem* **48**: 403–413.

Hopkins AL, Groom CR (2002). The druggable genome. *Nat Rev Drug Discov* **1**: 727–730.

Hu J, Hubbard SR (2005). Structural characterization of a novel Cbl phosphotyrosine recognition motif in the APS family of adapter proteins. *J Biol Chem* **280**: 18943–18949.

Irwin JJ, Shoichet BK (2005). ZINC—a free database of commercially available compounds for virtual screening. *J Chem Inf Model* **45**: 177–182.

Jacobsson M, Liden P, Stjernschantz E, Bostrom H, Norinder U (2003). Improving structure-based virtual screening by multivariate analysis of scoring data. *J Med Chem* **46**: 5781–5789.

Jones G, Willett P, Glen RC, Leach AR, Taylor R (1997). Development and validation of a genetic algorithm for flexible docking. *J Mol Biol* **267**: 727–748.

Jorgensen WL, Duffy EM (2002). Prediction of drug solubility from structure. *Adv Drug Deliv Rev* **54**: 355–366.

Kan D (2007). Studies of protein-ligand interactions and the discovery of new cyclophilin inhibitors. PhD Thesis, University of Edinburgh, Edinburgh.

Kitchen DB, Decornez H, Furr JR, Bajorath J (2004). Docking and scoring in virtual screening for drug discovery: methods and applications. *Nat Rev Drug Discov* **3**: 935–949.

Kofron JL, Kuzmic P, Kishore V, Colon-Bonilla E, Rich DH (1991). Determination of kinetic constants for peptidyl prolyl *cis–trans* isomerases by an improved spectrophotometric assay. *Biochemistry (Moscow)* **30**: 6127–6134.

Kontopidis G, McInnes C, Pandalaneni SR, McNae I, Gibson D, Mezna M *et al*. (2006). Differential binding of inhibitors to active and inactive CDK2 provides insights for drug design. *Chem Biol* **13**: 201–211.

Kothe M, Kohls D, Low S, Coli R, Cheng AC, Jacques SL *et al*. (2007). Structure of the catalytic domain of human polo-like kinase 1. *Biochemistry (Moscow)* **46**: 5960–5971.

Kramer B, Rarey M, Lengauer T (1999). Evaluation of the FLEXX incremental construction algorithm for protein–ligand docking. *Proteins* **37**: 228–241.

Lee JO, Russo AA, Pavletich NP (1998). Structure of the retinoblastoma tumour-suppressor pocket domain bound to a peptide from HPV E7. *Nature* **391**: 859–865.

Lipinski C, Hopkins A (2004). Navigating chemical space for biology and medicine. *Nature* **432**: 855–861.

Lipinski CA, Lombardo F, Dominy BW, Feeney PJ (1997). Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Adv Drug Deliv Rev* **23**: 3–25.

Liu Z, Sun C, Olejniczak ET, Meadows RP, Betz SF, Oost T *et al*. (2000). Structural basis for binding of Smac/DIABLO to the XIAP BIR3 domain. *Nature* **408**: 1004–1008.

McInnes C, Mazumdar A, Mezna M, Meades C, Midgley C, Scaerou F *et al*. (2006). Inhibitors of Polo-like kinase reveal roles in spindle-pole maintenance. *Nat Chem Biol* **2**: 608–617.

McInnes C, Mezna M, Fischer PM (2005). Progress in the discovery of polo-like kinase inhibitors. *Curr Top Med Chem* **5**: 181–197.

Moriguchi I, Hirono S, Liu Q, Nakagome Y, Matsushita Y (1992). Simple methods of calculating octanol water partition coefficient. *Chem Pharm Bull* **40**: 127–130.

Muegge I, Martin YC (1999). A general and fast scoring function for protein–ligand interactions: a simplified potential approach. *J Med Chem* **42**: 791–804.

Oprea TI, Allu TK, Fara DC, Rad RF, Ostopovici L, Bologa CG (2007). Lead-like, drug-like or 'Pub-like': how different are they? *J Comput Aided Mol Des* **21**: 113–119.

Oprea TI, Matter H (2004). Integrating virtual screening in lead discovery. *Curr Opin Chem Biol* **8**: 349–358.

Pauptit RA, Dennis CA, Derbyshire DJ, Breeze AL, Weston SA, Rowsell S *et al*. (2001). NMR trial models: experiences with the colicin immunity protein Im7 and the p85alpha C-terminal SH2-peptide complex. *Acta Crystallogr D Biol Crystallogr* **57**: 1397–1404.

Perola E, Walters WP, Charifson PS (2004). A detailed comparison of current docking and scoring methods on systems of pharmaceutical relevance. *Proteins* **56**: 235–249.

Pospisil P, Kuoni T, Scapozza L, Folkers G (2002). Methodology and problems of protein–ligand docking: case study of dihydroorotate dehydrogenase, thymidine kinase, and phosphodiesterase 4. *J Recept Signal Transduct Res* **22**: 141–154.

Qureshi SA, Kim RM, Konteatis Z, Biazzo DE, Motamedi H, Rodrigues R *et al*. (1999). Mimicry of erythropoietin by a nonpeptide molecule. *Proc Natl Acad Sci USA* **96**: 12156–12161.

Rarey M, Kramer B, Lengauer T, Klebe G (1996). A fast flexible docking method using an incremental construction algorithm. *J Mol Biol* **261**: 470–489.

Raymond JW, Jalaie M, Bradley MP (2004). Conditional probability: a new fusion method for merging disparate virtual screening results. *J Chem Inf Comput Sci* **44**: 601–609.

Rees DC, Congreve M, Murray CW, Carr R (2004). Fragment-based lead discovery. *Nat Rev Drug Discov* **3**: 660–672.

Rosenfeld R, Vajda S, DeLisi C (1995). Flexible docking and design. *Annu Rev Biophys Biomol Struct* **24**: 677–700.

Russ AP, Lampel S (2005). The druggable genome: an update. *Drug Discov Today* **10**: 1607–1610.

Scheufler C, Brinker A, Bourenkov G, Pegoraro S, Moroder L, Bartunik H *et al*. (2000). Structure of TPR domain–peptide complexes: critical elements in the assembly of the Hsp70–Hsp90 multichaperone machine. *Cell* **101**: 199–210.

Schuffenhauer A, Brown N, Selzer P, Ertl P, Jacoby E (2006). Relationships between molecular complexity, biological activity, and structural diversity. *J Chem Inf Model* **46**: 525–535.

Shave SR, Taylor P, Walkinshaw M, Smith L, Hardy J, Trew A (2008). Ligand discovery on massively parallel systems. *IBM Journal of Research and Development* **52** 1/2 January/March.

Singh J, Van VH, Liao Y, Lee WC, Cornebise M, Harris M *et al*. (2002). Identification of potent and novel alpha4beta1 antagonists using *in silico* screening. *J Med Chem* **45**: 2988–2993.

Stahl M, Rarey M (2001). Detailed analysis of scoring functions for virtual screening. *J Med Chem* **44**: 1035–1042.

Teague SJ, Davis AM, Leeson PD, Oprea T (1999). The Design of Leadlike Combinatorial Libraries. *Angew Chem Int Ed Engl* **38**: 3743–3748.

Todeschini R, Consonni V (2005). *Handbook of Molecular Descriptors*. Wiley-VCH: UK.

Vangrevelinghe E, Zimmermann K, Schoepfer J, Portmann R, Fabbro D, Furet P (2003). Discovery of a potent and selective protein kinase CK2 inhibitor by high-throughput docking. *J Med Chem* **46**: 2656–2662.

Vassilev LT, Vu BT, Graves B, Carvajal D, Podlaski F, Filipovic Z *et al*. (2004). *In vivo* activation of the p53 pathway by small-molecule antagonists of MDM2. *Science* **303**: 844–848.

Verdonk ML, Berdini V, Hartshorn MJ, Mooij WT, Murray CW, Taylor RD *et al*. (2004). Virtual screening using protein–ligand docking: avoiding artificial enrichment. *J Chem Inf Comput Sci* **44**: 793–806.

Vieth M, Hirst JD, Dominy BN, Daigler H, Brooks CL (1998). Assessing search strategies for flexible docking. *J Comput Chem* **19**: 1623–1631.

Wang JL, Liu DX, Zhang ZJ, Shan SM, Han XB, Srinivasula SM *et al*. (2000). Structure-based discovery of an organic compound that binds Bcl-2 protein and induces apoptosis of tumor cells. *Proc Natl Acad Sci USA* **97**: 7124–7129.

Wang R, Lu Y, Wang S (2003). Comparative evaluation of 11 scoring functions for molecular docking. *J Med Chem* **46**: 2287–2303.

Wang S, Meades C, Wood G, Osnowski A, Anderson S, Yuill R *et al*. (2004a). 2-Anilino-4-(thiazol-5-yl)pyrimidine CDK inhibitors: synthesis, SAR analysis, X-ray crystallography, and biological activity. *J Med Chem* **47**: 1662–1675.

Wang S, Wood G, Meades C, Griffiths G, Midgley C, McNae I *et al*. (2004b). Synthesis and biological activity of 2-anilino-4-(1H-pyrrol-3-yl) pyrimidine CDK inhibitors. *Bioorg Med Chem Lett* **14**: 4237–4240.

Woo HJ, Roux B (2005). Calculation of absolute protein–ligand binding free energy from computer simulations. *Proc Natl Acad Sci USA* **102**: 6825–6830.

Wu B, Li P, Liu Y, Lou Z, Ding Y, Shu C *et al*. (2004). 3D structure of human FK506-binding protein 52: implications for the assembly of the glucocorticoid receptor/Hsp90/immunophilin heterocomplex. *Proc Natl Acad Sci USA* **101**: 8348–8353.

Wu SY, McNae I, Kontopidis G, McClue SJ, McInnes C, Stewart KJ *et al*. (2003). Discovery of a novel family of CDK inhibitors with the program LIDAEUS: structural basis for ligand-induced disordering of the activation loop. *Structure* **11**: 399–410.

Wu X, Knudsen B, Feller SM, Zheng J, Sali A, Cowburn D *et al*. (1995). Structural basis for the specific interaction of lysine-containing proline-rich peptides with the N-terminal SH3 domain of c-Crk. *Structure* **3**: 215–226.

Xing L, Hodgkin E, Liu Q, Sedlock D (2004). Evaluation and application of multiple scoring functions for a virtual screening experiment. *J Comput Aided Mol Design* **18**: 333–344.

Yang JM, Chen CC (2004). GEMDOCK: a generic evolutionary method for molecular docking. *Proteins* **55**: 288–304.

Zhao L, Brinton RD (2005). Structure-based virtual screening for plant-based ERbeta-selective ligands as potential preventative therapy against age-related neurodegenerative diseases. *J Med Chem* **48**: 3463–3466.

Zheng CJ, Han LY, Yap CW, Ji ZL, Cao ZW, Chen YZ (2006). Therapeutic targets: progress of their exploration and investigation of their characteristics. *Pharmacol Rev* **58**: 259–279.