UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

# *In silico* synthesis of analogous lead libraries for drug design by molecular enumeration

by

Wolf Cochrane

Submitted in partial fulfillment of the requirements for the degree *Magister Scientiae*
in the faculty of Natural and Agricultural Sciences

Department of Biochemistry
School of Biological Sciences
Faculty of Natural and Agricultural Sciences
University of Pretoria
Pretoria
South Africa

April 2007

# Declaration

I, ..... declare that the thesis/dissertation that I hereby submit for the degree in ..... at the University of Pretoria has not previously been submitted by me for degree purposes at any other university and I take note that, if the thesis/dissertation is approved, I have to submit the additional copies, as stipulated by the relevant regulations, at least six weeks before the following graduation ceremony takes place and that if I do not comply with the stipulations, the degree will not be conferred upon me.

SIGNATURE..... DATE.....

# Acknowledgements

I would like to thank the National Bioinformatics Network for generously funding and supporting the LIGLIB project.

# Table of Contents

# List of Abbreviations

| | |
|---|---|
| ADT | AutoDock Tools |
| ASCII | American Standard Code for Information Interchange |
| CC | Combinatorial Chemistry |
| CDT | C/C++ Development Tools |
| EGEE | Enabling Grids for E-sciencE |
| FP | ferri/ferroprotoporphyrin IX |
| GSX | S-hexylglutathione |
| GSH | Glutathione |
| GST | Glutathione S-transferase |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| LGA | Lamarckian Genetic Algorithm |
| MCS | Maximum Common Substructure |
| MF | Molecular Fingerprints |
| MS | Mass Spectrometry |
| NMR | Nuclear Magnetic Resonance |
| PDB | Protein Data Bank |
| *Pf*GST | *Plasmodium falciparum* Glutathione S-transferase |
| RMS | Root Mean Square |
| RTI | Record Type Indicator |
| SGDD | Structure-Guided Drug Design |
| SLF | Scaffold, Linker and Fragment |
| SMILES | Simplified Molecular Input Line Entry System |
| SWIG | Simplified Wrapper and Interface Generator |
| VLTK | Virtual Library Tool Kit |
| WISDOM | World-wide In Silico Docking on Malaria |

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Pressing Issues in Drug Discovery

### 1.1.1 Costs Involved in Drug Discovery

Two recent studies have estimated the average cost of developing a new drug to be $802 million (DiMasi *et al.,* 2003) and $868 million (Adams & Brantner, 2006), discounted to year 2000 dollar value, increasing 7.4% above inflation each year. The main reasons for the high costs are the large number of potential drugs that go through to the expensive phases I, II and III stage, and fail because of poor ADMET (Absorption, Distribution, Metabolism, Excretion and Toxicity) properties. These false candidate drugs can be addressed by two approaches: firstly, developing and employing computational tools to predict ADMET properties, and calculating the probability of each candidate successfully passing the clinical testing stage; and secondly, making use of alternative techniques to Combinatorial Chemistry (CC) to search chemical space, techniques that are more prone to building drug-like molecules.

### 1.1.2 *In Vitro* Versus *In Silico* Techniques

The traditional *in vitro* method used in recent drug discovery is to synthesize a library through Combinatorial Chemistry and screen it with automated High-Throughput Screening (HTS) techniques. CC drug discovery emerged in the early 90's and generated immense expectations as it was believed at that stage that large, chemically diverse libraries would become the solution needed to search chemical space for drug-like molecules (Leach & Hann, 2000). In the late 90's it however became apparent that CC was not contributing to an

increase in the new lead discovery rate, and that the "blind" approach of unguided building of diverse libraries and using them as a universal screening set for all targets, was an expensive waste of time (Leach & Hann, 2000; Kubinyi, 2003). There were many reasons for the initial disappointment, the most obvious of which was the sheer vastness of chemical space. Some estimates of the drug-like chemical space range from $10^{18}$ (Martin, 1997) to $10^{60}$ (Bohacek *et al.,* 1996) possible molecules, far larger than can be explored in the reasonable future of humanity. Computational chemists realized that they needed to put more precombinatorial effort into the design of the libraries, moving away from diversity, rather focusing the search in localized chemical space. The only way of doing this is by working closely with biologists experienced with the target under investigation. The biologist can supply the combinatorial chemist with target structure constraints and active site pharmacophores, information that will give an indication to which combinatorial monomers have a chance of producing products capable of interacting with the target, thus focusing the chemical space search. Another major obstacle in CC is the poor ADMET nature of screening hit molecules, one of the major causes of the high discovery costs since these molecules fail during later, more expensive stages in the process. CC hits tend to be large molecules with poor bioavailability and pharmacokinetic properties. Here, the involvement of medicinal chemists is absolutely essential. They can help during the design phase, helping to select monomer building blocks that are drug-like and will probably result in drug-like products, and at the CC product investigation stage, where they can help with screening of hits, discarding the ones that are likely to have poor ADMET properties (Lombardino & Lowe, 2004). Typically, CC molecules are created while attached to a solid support such as a resin bead, which prevents polymerization and cross reactions, simplify purification, and makes total automation possible (Patrick, 2005). However, the rigid nature of the beads restricts the mobility of the product molecule, limiting the orientations it can probe a target structure with and leading to false negatives. Even though recent CC experiments have attempted to address the critical issues mentioned (Dolle *et al.,* 2006), there is still no notable new lead flux coming from CC technology, giving the impression that it is close to the maximum contribution that it can make, in isolation, to the drug discovery realm.

If a catch phrase has to be singled out in current drug discovery methodologies, it would definitely be "small fragment". In addition to the issues raised before, CC leads also tend to be large structures, and since lead refinement tends to make leads even larger, CC leads are likely to be non drug-like. An alternative approach would be to rather screen small molecule fragments, and to additively build leads from the resulting hits, giving smaller leads that are

already much more refined than CC leads. Instead of trying to find the end solution in one screen, the end solution is built up from smaller, positive screen results, thereby drastically reducing the chemical space searched (Figure 1.1). The approach, known as fragment-based drug discovery, is based on the notion that the free energy change resulting from fragments with weak binding affinities are additive when these fragments are linked together, thus resulting in leads with high binding affinities (Erlanson *et al.,* 2004). The fragments can be screened through high throughput Mass Spectrometry (MS), Nuclear Magnetic Resonance (NMR) or X-ray crystallography. HTS based on assays are not viable since it is probable that many fragments will be missed because of their weak binding affinity. MS can be extremely high throughput, but is limited in the amount of structural information it can give on the compound-target binding (Swayze *et al.,* 2002). NMR was the first screening method used to gather structure-activity information for compounds (Moore *et al.,* 2004), but it is the recent implementation of X-ray crystallography which is giving the best structure-activity and binding site position information. Knowledge of the precise fragment binding site greatly helps the process of designing inter-fragment linker scaffolds as the exact positions of the atoms to be attached are known. The crystallography approach has been implemented successfully on various occasions (Blundell *et al.,* 2002; Carr & Jhoti, 2002; Nienaber *et al*., 2000). Crystals are soaked with mixtures or single molecules, and then put through automated high throughput X-ray diffraction at a synchrotron (Blundell & Patel, 2004). The X-ray data of the ligand-protein complex is solved by automatic methods, such as Autosolve, and then compared to the original uncomplexed protein structure, thus solving the ligand-protein interaction. The fragment-based approach to drug discovery has a lot of potential, but is important to remember that for any given experiment, the results can only be as good as the quality of fragments used for the screening, and the quality of the linkers used to link them.

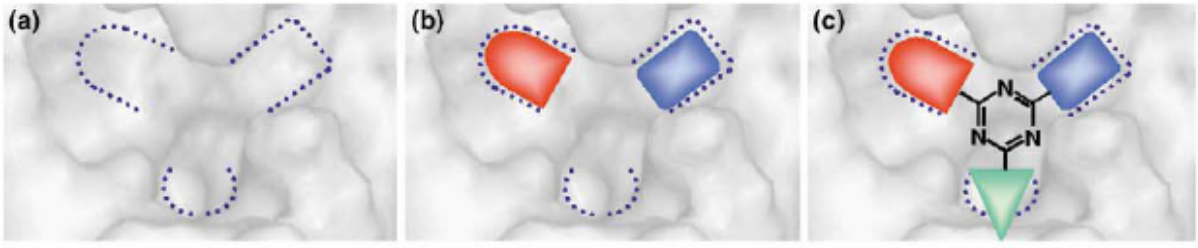



Figure 1.1: a & b) The process screens for small fragments that bind small cavities within the active site. c) The fragments identified to bind specific cavities are linked together by an appropriate linker fragment. Adapted from Blundell & Patel (2004).

The computational power of modern computers and the tools and algorithms being developed for them have led to *in silico* techniques making a major contribution to drug discovery, and since the more data is used, the more data can be generated, it can reasonably be expected to do so even more in the near future. The main driving forces behind *in silico* drug discovery are the relative low cost of working *in silico* versus working *in vitro*, the new levels of accuracy and functionality provided by software and algorithms, and the quantity of data available to work on.

The most obvious and general *in silico* contribution to drug discovery comes in the form of chemoinformatics. Research in the information age has led to the generation of massive amounts of data, which needs to be stored, managed, integrated and queried sufficiently, so that it can contribute to further information discovery.

Since the first major modern study of drug-like molecules and their characteristics was done (Lipinski *et al.,* 2001), *in silico* ADMET property prediction techniques have become increasingly accurate (Hunter, 2007) and holds the biggest potential to contribute to the drug discovery process. There is a direct correlation between cost and the amount of time a structure spends in the drug discovery process, and therefore, because poor ADMET properties is the single largest reason for later stage failure (van de Waterbeemd & Gifford, 2003), a “fail early, fail cheap” approach is the best way of cutting drug development costs. ADMET models are built from experimental pharmacokinetic data and corresponding molecular descriptors (Colmenarejo, 2005). The models are used to: bias which fragments should be used for library generation, *in silico* and CC; suggest ways of improving poor pharmacokinetic properties of leads; and for screening existing compound libraries for molecules likely to have poor pharmacokinetic properties.

The ever increasing number of known protein structures has given rise to the parallel growth of the structure-guided drug design (SGDD) approaches, as can be seen in the review by Hardy & Malikayil (2003) where the authors report forty drugs that have reached clinical trials. In SGDD, ligand molecules are built through various different approaches, all guided by the molecular structure of a target macromolecule (Ghosh *et al.,* 2006). SGDD library building techniques are discussed in Section 1.3. Once a set of molecules have been designed, they undergo virtual screening through either molecular docking or pharmacophore screening. Molecular docking predicts the binding conformation and energies of a given ligand (Kellenberger *et al.,* 2004), while pharmacophores are sets of features in 3D, defined as the complement of a protein active site, and are used to screen ligands to find structures that fit well onto the feature sets (Griffith *et al.,* 2005). SGDD has also been taken further by

the emergence of high-throughput virtual screening where massive datasets are screened against macromolecules (Mizutani & Itai, 2004). SGDD is totally dependent on *in silico* technologies as they are the only efficient way of investigating macromolecule structures, designing structure libraries and testing ligand-macromolecule interactions.

When comparing the pros and cons of the existing drug discovery technologies, it is believed that there is no single technology that can be the answer, but rather that the synergistic use of all the technologies combined will give the best possible solution to current drug discovery. The diverse nature of CC can be combined with the small molecule approach and SGDD (Rupasinghe & Spaller, 2006), perhaps taking natural occurring structures into account (Muller *et al.,* 2006; Dekker *et al.,* 2005; Samiulla *et al.,* 2005), giving diverse libraries focused in relevant chemical space. These libraries can be filtered further by *in silico* ADMET screening, resulting in high quality lead molecules with high probabilities of their refined candidates making it through clinical testing.

### 1.1.3 Open Source and Drug Discovery

Unfortunately, the recent growth in *in silico* techniques and publications have not been followed by the relative growth in the availability of open source/free drug discovery tools, as is evident when comparing chemoinformatics with bioinformatics which has made major leaps by successfully implementing the open source paradigm. The pharmaceutical companies have become dependent on extremely expensive software from vendors such as Accelrys, Tripos and Openeye; and the in-house redundant individual development of massive overall drug discovery systems (Feuston *et al.,* 2005). On the other hand, the academic community has very little access to expensive proprietary vendor tools, and simply does not have the resources required to build large discovery systems, thereby being severely limited in the size of the focus area in which they can perform discovery. Arguably, the biggest cause for the slow chemoinformatics open source development is the strong demand for software from the pharmaceutical companies, backed by immense financial resources, making the opportunity cost of developing open source tools very high. The issue is not just limited to open source development, but also the availability of open data. The solution to the problem is to educate the role players about the many advantages of open source (DeLano, 2005), both to big pharma and academia, and indeed its seems as if the paradigm is starting to change. There are currently a fair amount of good open source tools becoming available

(Geldenhuys *et al.,* 2006), and the emergence of the Blue Obelisk initiative – which promotes open software, open standards and open data – is a definite step in the right direction (Guha *et al.,* 2006).

## 1.2 Rational Drug Design Process

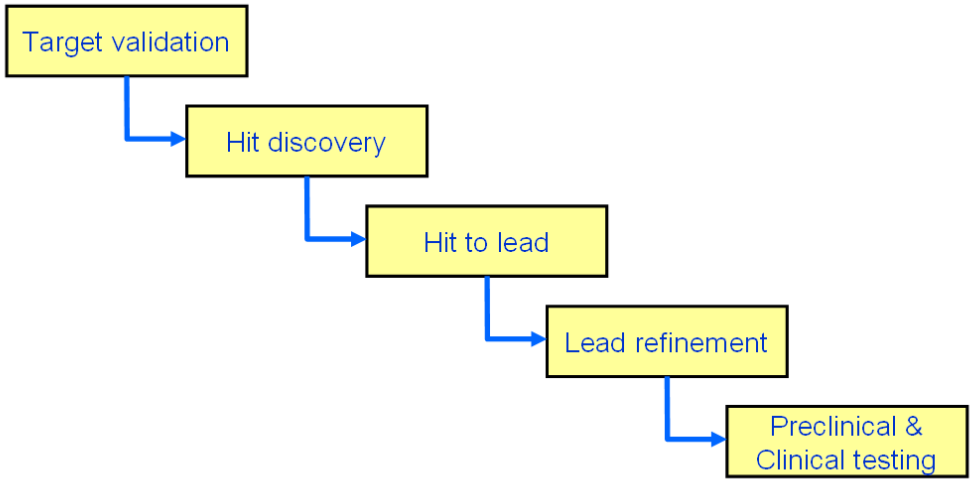



Figure 1.2: Simple representation of the steps in rational drug design.

Even though there are various approaches to rational drug design, the higher level one represented in Figure 1.2 captures the main steps that each approach undergoes. The first step, target validation, is extremely important as drug discovery on an ineffective target is an expensive waste of money and resources. When working on diseases caused by microorganisms or viruses, a target is sought that will, if prohibited from performing its task, have a detrimental effect on the attacking organism. For proteins, this could be: interference with carrier proteins, preventing them from moving critical polar molecules over the cell membrane; exposure of receptors to an agonist or antagonist, thereby manipulating down stream signal transduction of cells; and inhibition of enzymes, disrupting critical metabolic pathways. Target identification and validation is performed by a wide range of technologies including, genomics, microarrays, proteomics, systems biology, gene knockouts, and bioinformatics (Williams, 2003).

The second step, hit discovery, is where the search for drugs begins. During a knowledge-based process, the portions of chemical space to be investigated are decided on.

Once the chemical space has been selected, a hit library has to be generated/selected, again making use of available knowledge of the target and initial screens. The idea is to select as small as possible chemical space area, while building a screening library as diverse as possible within it. A range of tools are available that can assist in the library generation process and they are discussed in the following section.

The third step, hit to lead, typically involves screening and filtering. There is a fair amount of controversy regarding when a molecule moves from being a hit, to becoming a lead. For this purpose, it is said that hits are the molecules that make it through the initial library screening stages, giving a relatively large set, and leads are in the much smaller set that have made it through the later, more stringent screening steps. The hits from step 2 have already given insight into which fragments can and cannot be used, further decreasing the chemical space of interest. Using the tools originally used for the hit generation, a more focused hit library can be generated, investigating around the areas of interest indicated by the initial hit library. This can be done iteratively, with each new library screened under more stringent filters, each iteration resulting in a smaller, more focused set of molecules until finally, only a small set of molecules remains, becoming leads.

Even though medicinal chemists should be involved in all the steps of drug discovery, it is in step 4 that their skills are most required. They make small changes to leads to optimizing bioavailability and distribution qualities, and minimizing potential toxicity. It is the medicinal chemist's responsibility to make sure that only excellent lead candidates are allowed through to the next step, as it is by far the most expensive portion of the discovery process and has an extremely high failure rate.

Step 5 is the final two stages of discovery, preclinical and clinical testing. In preclinical testing the handful of refined leads are tested in assays and animal models. After successful preclinical testing, typically a single drug candidate is sent through to clinical testing. Clinical testing is made up of three phases. The first phase is done on a small number of healthy patients and is used to test *in vivo* ADMET properties of the candidate. The second phase is done on patients that have contracted the disease, and is done to further test safety and the efficacy of the candidate in treating the disease. In the third phase, patients, possibly numbering in the thousands, are screened to further test the efficiency and the possible occurrence of less common side effects.

## 1.3 Building *In Silico* Hit Molecule Libraries

There are many *in silico* techniques that are well suited to hit library generation. When searching for hits, the chemical space frameset is very large and it is not yet worthwhile to invest in production and screening of large number of hit structures. The following sections discuss the three main *in silico* approaches that can be used to narrow down the portion of chemical space of interest. Each of the techniques has advantages and disadvantages, of which the disadvantages can to a certain extent be addressed by developing integrated systems using a range of different techniques, each covering the other's shortcomings.

### 1.3.1 *De Novo* Molecule Building

The term "*de novo*" means new or from scratch, and molecules generated through *de novo* techniques are exactly that, new, novel structures that are generated automatically within a set of constraints, typically a target active site. There are four main considerations when designing a *de novo* tool: active site constrains; building technique; scoring mechanism; and overall implementation algorithm (Gillet *et al.,* 1994).

For the molecules being generated to be relevant, they are built within a constraint set made up of the spatial constraints and the protein interaction sites. Rule-based methods of constraint deduction find interaction positions within the active site and maps it onto a pharmacophore, while grid based methods build a grid map of the active site and calculates energy potential at each grid point using probe atoms. GRID (Goodford, 1985) is a popular tool used by *de novo* tools to do the grid calculation, although, many tools use their own grid implementations to perform the calculation (Bywater *et al.,* 2004).

There are two types of building blocks used to build the molecules: atoms and fragments. Atoms can lead to very diverse molecules, but this is also its major drawback because it is not a viable solution to searching the vast chemical space. Molecules built up from atoms are also not very drug-like. Using fragments drastically reduces the search space and if a good fragment set is used, will result in molecules that are more drug-like. The latter is the building block of choice with virtually all the new tools of recent years using it (Schneider & Fechner, 2005). The building-up of fragments is done by either the growing approach or the linking approach. With the growing approach, a single seed fragment is placed at an interaction position, and the molecule grown from this fragment. The linking approach starts

with a set fragments placed at various interaction positions, and are simultaneously grown until they link up with each other.

The design process can generate many different molecules and it is necessary to distinguish between the good and the bad ones by scoring them. The scores also give an indication of which fragments contribute to high scores and which do not. There are many scoring techniques available and all of them judge molecules by estimating the binding free energy change. Some tools go beyond binding free energy scores and also incorporate some ADMET filters (Wang *et al.,* 2000).

The sheer vastness of possible solutions does not allow for *de novo* design to be performed indefinitely. There has to be an algorithm to guide the process and to decide how the building should take place. Three typical algorithmic approaches are combinatorics searches, Monte Carlo searches and evolutionary algorithms. Fragments common in high scoring molecules are likely to contribute to high scores and are allowed to be used in subsequent runs.

*De novo* design approaches can rapidly generate very diverse libraries, although they are prone to be non-drug-like. The technique is also criticized for not having any human input in the process and because it cannot be guided by biological knowledge.

### 1.3.2 Screening Chemical Databases

A well established method of generating hit libraries is to screen existing chemical databases for structures similar to known ligands and inhibitors of a specific target. The approach is based on the similarity principle that states that structurally similar molecules are more likely to have similar biological activities. The principle was validated by a study done by Martin *et al.* (2002) in which they found that structures compared with Daylight fingerprints that have a Tanimoto similarity coefficient >84.9 have a 0.3 probability of having the same activity.

The biggest hurdle to the approach has been the availability of screening databases and it is only recently that international efforts were made to counter it. Traditionally, most of the useful databases such as the Available Chemicals Directory (ACD, http://cds.dl.ac.uk/cds/datasets/orgchem/isis/acd.html) and the American Chemical Society's Chemical Abstract Service (CAS, http://www.cas.org/), have been in proprietary hands and could only be accessed through expensive licensing (Jónsdóttir *et al.,* 2005). Recently-created

large public repositories include PubChem (http:// pubchem.ncbi.nlm.nih.gov), ChemBank (Strausberg & Schreiber, 2003), ChemDB (Chen *et al.,* 2005), ZINC (Irwin & Shoichet, 2005) and ChEBi (http://www.ebi.ac.uk/chebi) but they have met with fierce legal resistance from the American Chemical Society (Marris, 2005). Making more chemical structures available is only the first step; new and innovative techniques are also required to search the databases quickly and effectively.

Chemical databases vary in their size, composition and data sources. Databases such as the World Drug Index (WDI, http://scientific.thomson.com/products/wdi/) and DrugBank (Wishart *et al.,* 2006) contain drug molecules and their properties, and should be the first databases searched as it might be possible that an existing drug can act on a new target of interest. The Human Metabolome DataBase (HMDB, http://www.hmdb.ca/) and FooDB (http://hmdb.med.ualberta.ca/foodb) contain metabolite, and food and additive structures, respectively, and are a good source of structures with good ADMET properties. PubChem is fast becoming the Chemoinformatics equivalent of the bioinformatics database, GenBank. PubChem Substance currently contains more than 17 million structures collected from 53 databases (http://pubchem.ncbi.nlm.nih.gov/sources/sources.cgi).

There are two main approaches to searching for similar structures, Molecular Fingerprints (MF) and Maximum Common Substructure (MCS). MF are binary strings with every bit in the string indicating the presence or absence of specific molecular attributes, mostly structural (Haigh *et al.,* 2005). The bits typically represent the presence or absence of certain substructures, rings, ring systems and reactive groups. The similarity of two molecules is calculated by comparing their molecular fingerprints and calculating a score using an association based coefficient that takes into account the number of bits common to two bit-strings. The Tanimoto coefficient is currently the most popular of the coefficients, but two other coefficients, the Forbes coefficient and the Russel/Rao coefficient, are also gaining popularity (Willett, 2003). Molecular fingerprint techniques perform very basic logic operations with very low computational overhead and are extremely efficient when it comes to performing similarity searches on large molecular databases. MCS techniques measure similarity by finding the largest substructure common to two molecules. It is a graph-based method that makes use of complex isomorphism algorithms (Raymond & Willett, 2002) such as reduced graphs (Harper *et al.,* 2004) and the maximum clique detection in line graphs (Raymond *et al.,* 2002). The MCS approach is an excellent method of finding true similar structures but is computationally intensive.

Currently available computing power now also allows starting with the target itself, docking extremely large molecular datasets against it. An example of such an approach can be found in the WISDOM (World-wide In Silico Docking on Malaria) project. WISDOM is a virtual screening pipeline run on the EGEE (Enabling Grids for E-sciencE) grid platform (Gagliardi *et al.,* 2005), and was specifically developed for drug discovery in traditionally neglected diseases such as malaria (Jacq *et al.,* 2006). The first run took place in 2005. Malaria proteins were targeted and 41000 compounds docked in an eighty year CPU run, which took only six weeks on the EGEE grid.

The wide range of different types of databases allows scientists to apply different drug design approaches by targeting specific databases. The power of the approach only gets stronger as the number of databases and structures increase. The approach's biggest potential lies in that it can be an excellent tool for finding more information on a target and its interaction with ligands, enabling the design scientist to make more informed decisions, and can work synergistically with the other techniques discussed here.

### 1.3.3 Fragment-Based Enumeration

Fragment-based enumeration is distinguished from fragment-based *de novo* techniques by the fact that the enumeration takes definite inputs from the user regarding scaffold, permutation positions and fragments, while *de novo* techniques make use of algorithms to determine these parameters.

The typical fragment-based enumeration techniques make use of a Markush structure approach, which permutes over a range of fragments at specific functional groups on a core scaffold. A Markush structure is a core molecule structure with variation sites on it labelled as functional groups (R-groups), each of which has a range of fragments which can be permuted over (Leland *et al.,* 1997). The library is built by creating a new molecule for each of the possible combinations of fragment permutations over all the R-groups on the scaffold. A range of tools (SMILIB, Legion, CombiLibMaker, virLibLinker, SLF_LibMaker, VLTK) have already been developed to perform fragment-based enumeration but the only tool freely available is SMILIB (Schüller *et al.,* 2003). Legion and CombiLibMaker are proprietary tools developed by Tripos (http://www.tripos.com), while virLibLinker (Liao *et al.,* 2005) and SLF_LibMaker (Krier *et al.,* 2005) are not available for download. VLTK (Virtual Library

Tool Kit) was developed by Merck as an in-house tool for designing, enumeration, optimizing and tracking compound libraries, and is only available on the Merck intranet.

All recent implementations of fragment-based enumeration make use of the scaffold, linker and fragment (SLF) approach. Molecular enumeration with the SLF approach takes place around a non-variable core scaffold structure. The permutations during enumeration take place at predefined permutation positions – usually hydrogen atoms – on the scaffold. At each permutation position a predefined set of linkers and fragments are permuted over. The user will select a scaffold structure relevant to the protein active site, select permutation sites on the scaffold in proximity of a protein interaction site, select fragments (cations, anions, hydrogen bond acceptors, hydrogen bond donors, lipophilics) that could possibly take part in interactions at the interaction site, and select a range of linkers to connect the fragments with the permutation site selected on the scaffold. The role of the linker is to vary the distance between the protein interaction site and the complimentary fragment at the end of the linker.

SMILIB, CombiLibMaker and virLibMaker perform the molecular enumeration with the linkers, fragments and core structure in SMILES (Simplified Molecular Input Line Entry System) format (Weininger, 1988). SMILES is a linear string notation similar to natural language and represent molecules as 2D-graphs. The output library generated by the tools is also in SMILES format, which means that another tool has to be used to transform the library into 3D structures. Two proprietary tools that can do the 3D transformation are Concord and Corina.

Most enumeration tools make use of added syntax in the SMILES to indicate attachment and variable points in the structures, which means that the user will have to type the input scaffold, linkers and fragments manually. The library is built by recursively inserting linkers, fragments and the scaffold SMILES into the appropriate positions indicated by the special attachment syntax at the various string positions.

Fragment-based enumeration is an excellent method of searching local chemical space around a hit molecule. A good starting hit-molecule would be to find molecules known to bind to the protein target of interest, proteins from the same family as the target, or that are similar to ligands known to bind to the protein target or to members of its family. The chemical space around the hits is searched by using the core of hit molecules as scaffolds and making permutations at specific points on it. These hit scaffolds are good positions to start a search in chemical space since the scaffolds are known to fit into the target active site, positions fragments at the correct ligand-target interaction sites and because the scaffold

might also interact with the target active site. The user can generate large chemical libraries or small focused ones by varying the number of permutation position, linker and fragment.

There are already a number of implementations of the fragment-based enumeration technique; however, there was, before the onset of this project, no easy to use open source version available to the scientific community. The main advantage of fragment-based enumeration is that it allows the scientist to directly apply knowledge to the design process, but, therein lies its biggest obstacle; it is only as effective as the input given by the scientist.

## 1.4 Brief Overview of LIGLIB and the Project

LIGLIB is an open source molecular enumeration tool that builds analogous chemical libraries using a hit chemical compound as a starting scaffold. LIGLIB performs the molecular enumeration with an SLF Markush-structure approach described above. LIGLIB is a practical, easy to use tool with an intuitive user interface that allows users to easily build chemical libraries fast. LIGLIB was built as an extension for the molecular graphics program, Chimera (Pettersen *et al.,* 2004), which was developed at the University of California at San Francisco (UCSF). Chimera's molecular visualization capabilities are exploited by LIGLIB to allow the user to input a 3D chemical scaffold instead of typing out a large scaffold in SMILES format.

With other open/free tools the user would have indicated permutation position on the scaffold with special syntax in the scaffold SMILES, but with LIGLIB permutation sites are indicated by simply clicking on the atoms of the input scaffold where the sites are to be located. For each permutation site, a separate set of linkers and fragments are selected by the user. LIGLIB has a linker/fragment manager tool, which the user can use to build, manage and store linker and fragment libraries. The fragments and linkers are in SMILES format but do not require any special syntax as other enumeration tools do.

The actual building of the chemical library is done with the scaffold, fragments and linkers in SMILES format, which requires the 3D scaffold to be converted into SMILES. An algorithm was developed to map the permutation sites selected by the user on the 3D structure onto the same atoms of the scaffold in SMILES format, using the chemical connectivity and graph isomorphism. The initial output library generated by LIGLIB is in SMILES format. LIGLIB has an interface for the 3D generation tool Corina and if it is installed locally on the user's computer, it will be used to also generate a 3D structure library.

LIGLIB is a versatile tool and can be implemented at various stages of the rational drug design process discussed earlier. Its main application will be in the hit discovery stage where it will be used to build relatively large libraries to be screened for hit. It is also a useful tool in discovering leads as it can be used to further search the chemical space around initial hits.

To illustrate how LIGLIB can be implemented, and also to validate it, a complete *in silico* drug design experiment was performed in designing a hit library for *Plasmodium falciparum* Glutathione S-Transferase (*Pf*GST) using LIGLIB. *Pf*GST was investigated in literature and validated as drug target. Different strategies were hypothesized to target *Pf*GST. The library was generated with a reduced structure of glutathione as a scaffold, fragments and linkers designed based on knowledge from the *Pf*GST active site, and the observed interactions between *Pf*GST and S-HexylGlutatione (GSX), which was observed in the crystal structure of *Pf*GST with the inhibitor. The resulting library was screened by molecular docking with AutoDock, and the results investigated and explained in the context of the successful implementation of LIGLIB and the fragment-based enumeration approach it uses.

## 1.5 Goals

The project had three main goals to achieve:

- Develop a method that will efficiently build ligand libraries with the Markush structure molecular enumeration technique (Chapter 2).
- Design and develop a software package that implements the devised method of library building (Chapter 3).
- Validate the software and the Markush structure enumeration technique by using the software in an actual drug discovery experiment targeting *Plasmodium falciparum* Glutathione S-transferase (Chapter 4).

# Chapter 2

# Methodology Employed by LIGLIB

## 2.1 Introduction

Chapter 2 describes the methodology built into LIGLIB that generates the library of analogous chemical structures. The process can be split up into four main sequential steps: Gathering user input; converting input scaffolds into SMILES format; building the library and converting the library into 3D mol2 format. The user input includes a scaffold structure in mol2 format and linkers and fragments in SMILES format. The library generation takes place in SMILES format, which means that the input scaffold should be converted to SMILES. A graph isomorphism algorithm is implemented to map the same atom in the SMILES and mol2 structures to each other, which allows LIGLIB to find in the 2D SMILES structure the atom selected by the user on the 3D structure in Chimera. The library generated by LIGLIB is given as output in both 2D SMILES and 3D mol2 format.

### 2.1.1 Molecular Structure Representation

#### 2.1.1.1 Tripos Mol2 Format

Tripos Mol2 files are ASCII files that represent molecules in 3D with a large amount of molecular attribute data (http://www.tripos.com). There are 32 different attribute types that can be used in mol2 format and each is located in a unique subsection in the file (Appendix A). Each attribute section is identified in the mol2 file by a unique Record Type Indicator (RTI), which is a line starting with "@", followed by the attribute name and ending with an end of line character. An attribute section is terminated when a new RTI or the end of file is

found. The two attribute sections relevant to LIGLIB's methodology are the atom and bond sections.

The atom section is identified with the RTI "@<TRIPOS>ATOM". Each line in the atom section contains all the information to create an atom in a molecule. The first column entry in the line is the atom_id, which is an integer used to reference atoms. The second entry is the atom_name and is a string used to name the atom. The next three entries are the x, y and z coordinates of the atom. The sixth entry in the line is the atom_type variable of the atom, which has a SYBYL atom type value relevant to the atom represented by that line. The other column entries of an atom entry are not relevant to LIGLIB and are not discussed here.

The bond section is identified with the RTI "@<TRIPOS>BOND". Each bond in the molecule is represented by a single line in the bond section. The first entry in a bond line is the bond_id and is an integer used to reference the bond. The second and third entries are atom_id's and indicate the atoms that are connected by the current bond. The fourth entry indicates the bond type.

#### 2.1.1.2 SMILES Format

Weininger (1988) developed SMILES (Simple Molecular Input Lines Entry System) as an efficient and intuitive notation to work with and store chemical structures *in silico.* SMILES structures are linear strings of characters that represent chemicals as graphs, with the atoms represented as vertices and the bonds between them as edges. The intuitive nature and simple rules allow a novice user of the notation to easily and accurately create his/her own molecules in SMILES. The Rules relating to atoms, bonds, branches and cyclic structures in SMILES are discussed below.

**(1) Atoms.** Atoms are represented by their atomic symbols enclosed in square brackets with normal atoms starting with uppercase and aromatic atoms with lowercase letters (Table 2.1). The number of hydrogens attached to the atom, and their charge have to be specified within the square brackets. The number of attached hydrogens are indicated by "H" followed by a digit indicating the number of hydrogens if more than one is present. The charge is indicated by "+" and "-" symbols. If the charge is more than one it is indicated by sequential positive or negative symbols to the same number of the charge, or by a single "+" or "-" symbol followed by the number of the charge. Atoms from the organic subset, B, C, N, O, P,

S, F, Cl, Br and I can be written without the brackets if the number of attached hydrogens conform to the lowest normal valence consistent with explicit bonds.

Table 2.1: Examples illustrating the SMILES rules relating to atoms

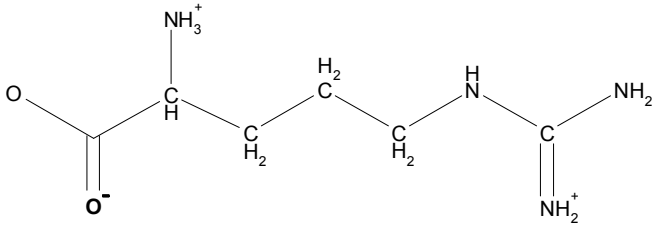
| SMILES String | Description |
|---|---|
| [OH2] | Water ($H_2O$) |
| [OH-] | Hydroxyl anion |
| [OH3+] | Hydronium cation |
| [Fe+2] | Iron (II) cation |
| [Fe+++] | Iron (III) cation |
| C | Methane ($CH_4$) |

**(2) Bonds.** The SMILES notation of single, double, triple and aromatic bonds are "-", "=", "#" and ":", respectively (Table 2.2). If two atoms are adjacent to each other and no bond is indicated, it is assumed that bond is single or aromatic, depending on whether the atoms are indicated to be aromatic or not. Disconnected structures – that are not connected by a covalent bond – are indicated by separating their SMILES strings with a period.

Table 2.2: Examples illustrating the SMILES rules relating to bonds between atoms

| SMILES String | Description |
|---|---|
| C-C | Ethane ($CH_3CH_3$) |
| C=C | Ethylene ($CH_2$=$CH_2$) |
| CO | Formaldehyde ($CH_2$=O) |
| [CH3][CH2][OH] | Ethanol ($CH_3CH_2OH$) |
| [NH4+].[Cl-] | Ammonium Cloride ($NH_4Cl$) |

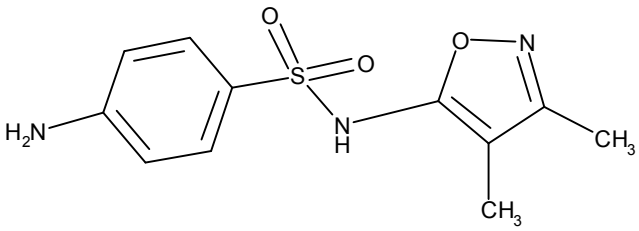
**(3) Branches.** Branches in a chemical are indicated by placing the branched substructure in parentheses. Sub-branches can be nested in higher-level branches, while consecutive branches can be stacked sequentially and indicate multiple branches at one atom position (Figure 2.1).



[CH]([C]([OH])=[O])(=[NH2])[CH2] [CH2] [CH2][NH]([NH])

Figure 2.1: A SMILES-string representing Arginine. Branches can be nested and stacked.

**(4) Cyclic structures.** Cyclic structures are represented by breaking a bond between two atoms in the cyclic structure and labeling those ring closure atoms at the bond position with a digit immediately after the atom symbols (Figure 2.2). If an Atom takes part in more than one ring closure, those ring closures are indicated by sequentially adding closure digits to the end of the atom symbol. Double-digit closures have a "%" directly to the left of the digits in the string in order to distinguish them from sequential single digit closures. Aromatic structures are indicated by lowercase atom symbols, but SMILES interpreters will also recognize the Kekulé form. For example, benzene should be written as c1ccccc1 in SMILES but C1=CC=CC=C1 will also be interpreted as aromatic.



[cH]1[cH][c]([cH][cH][c]1[NH2])[S]([NH][c]2[c]([c]([n][o]2)[CH3])[CH3])(=[O])=[O]

**or**

[cH]%91[cH][c]([cH][cH][c]%91[NH2])[S]([NH][c]%81[c]([c]([n][o]%81)[CH3])[CH3])(=[O])=[O]

**or**

[cH]1[cH][c]([cH][cH][c]1[NH2])[S]([NH][c]1[c]([c]([n][o]1)[CH3])[CH3])(=[O])=[O]

Figure 2.2: The structure of 4-Amino-N-(3,4-dimethylisoxazol-5-yl)benzenesulfonamide and three SMILES strings all representing the molecule. The top string make use of "1" and "2 " as ring closures. The middle string illustrates how double digits can be used as ring closures by adding a "%" in front of it. The bottom string illustrates how the same ring closure digit can be reused.

### 2.1.2 Graph Theory

The underlying principles of graph data structures are very simple and can be easily understood. A graph is made up of two basic elements: **Vertices** and **Edges**. Vertices, or nodes, are the different data instances to be represented, and edges are the connections between these data instances. Graphs can thus be used to represent data instances with the relevant relationships between them. It is not hard to see how graphs can be used to represent molecules. The most basic implementation is to make atoms, vertices, and the bonds, edges.

The term **Graph Theory** is used to describe the mathematics relating to graph networks. The following terminology can be used to describe graphs mathematically. A graph G is made up of a set of vertices V(G) and a set of edges E(G). An edge connecting two vertices, $v_i$ and $v_j$, will be denoted by E($v_i$ ,$v_2$) Є E(G), where $v_i$ Є V(G) and $v_j$ Є V(G).

Graphs G = (V, E) and G' = (V', E') are said to be **isomorphic** if there exists a bijection function f:V -> V' so that xy Є E <=> f(x)f(y) Є E' for all x,y Є V. In other words, there is a one to one correspondence between the vertices of the graphs with all the edges in both graphs existing only if it exists between the same two vertices in the other graph.

### 2.1.3 LIGLIB Vector Application

LIGLIB uses UCSF Chimera as a vector application (Pettersen *et al.,* 2004). Chimera is a free tool for academics and nonprofit researchers and was designed with extensibility as the primary goal. Chimera's design is divided into two levels, core and extensions. The core level provides basic services and molecular visualization while the extensions level provides higher-level functionality. Time critical operations performed by the core are handled in a C++ layer while all other core functions are handled in a Python layer. All significant objects and functions from the C++ layer are made available to the Python layer and can be easily accessed by extension software. Chimera is well-documented from both a user and developers' perspective, and all documentation can be accessed online (http://www.cgl.ucsf.edu/chimera/docs).

Chimera is released with Python and IDLE imbedded into it, which serves as an extension-building environment for users and developers. The extensions can be written entirely in Python or with a combination of Python and C/C++. The Chimera user interface was developed with the Python Tkinter module, which is an interface to the Tk GUI toolkit,

and extensions to the Chimera interface should also be made with the Tkinter tool. Extensions are packaged in Python packages and are imported to Chimera using the Preferences/Tools dialog, with which the user simply adds the package directory to the tools location file paths. The extension becomes available under the Tools menu section and will run when selected from it.

Nearly all elements of Chimera are represented as Python objects and are available to developers when coding extensions. The most important objects of Chimera are the atoms, bonds, residues and molecules, and these objects are related to one another in a class hierarchy (Figure 2.3).

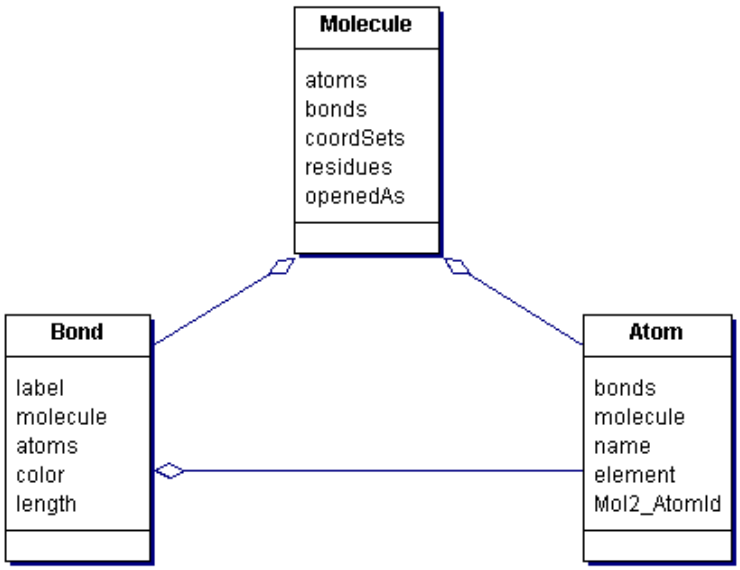



Figure 2.3: Class diagram of Molecule, Bond and Atom objects as they appear in Chimera. Only some of the attributes are listed for each object. The hierarchy of the objects is indicated by the lines (adapted from http://www.cgl.ucsf.edu/chimera/docs/ProgrammersGuide/Examples/index.html).

## 2.2 Methodology

### 2.2.1 Input from the User

LIGLIB needs four user inputs to generate the chemical library: a 3D scaffold; permutation positions on the scaffold; linkers and fragments activated for each permutation position. The inputs are grouped into sets, one for each permutation position. Each set has a permutation position, a range of activated linkers and a range of activated fragments. These

input sets are further referred to as permutation instruction sets. All user inputs are gathered by the intuitive dialog windows of the LIGLIB front-end interface imbedded in UCSF Chimera. One of the most important distinguishing factors of LIGLIB over the only other freely available enumeration tool, SMILIB, is the ease with which it can be used. There is no need for manual file editing or cumbersome command line interfaces. All inputs are gathered systematically from the user. The main LIGLIB dialog window opens when the user is building a library, the Build Library window, allows the user to add, remove and visualize existing permutation instructions, which means that the user can dynamically manage the LIGLIB instruction input process (Figure 2.4).

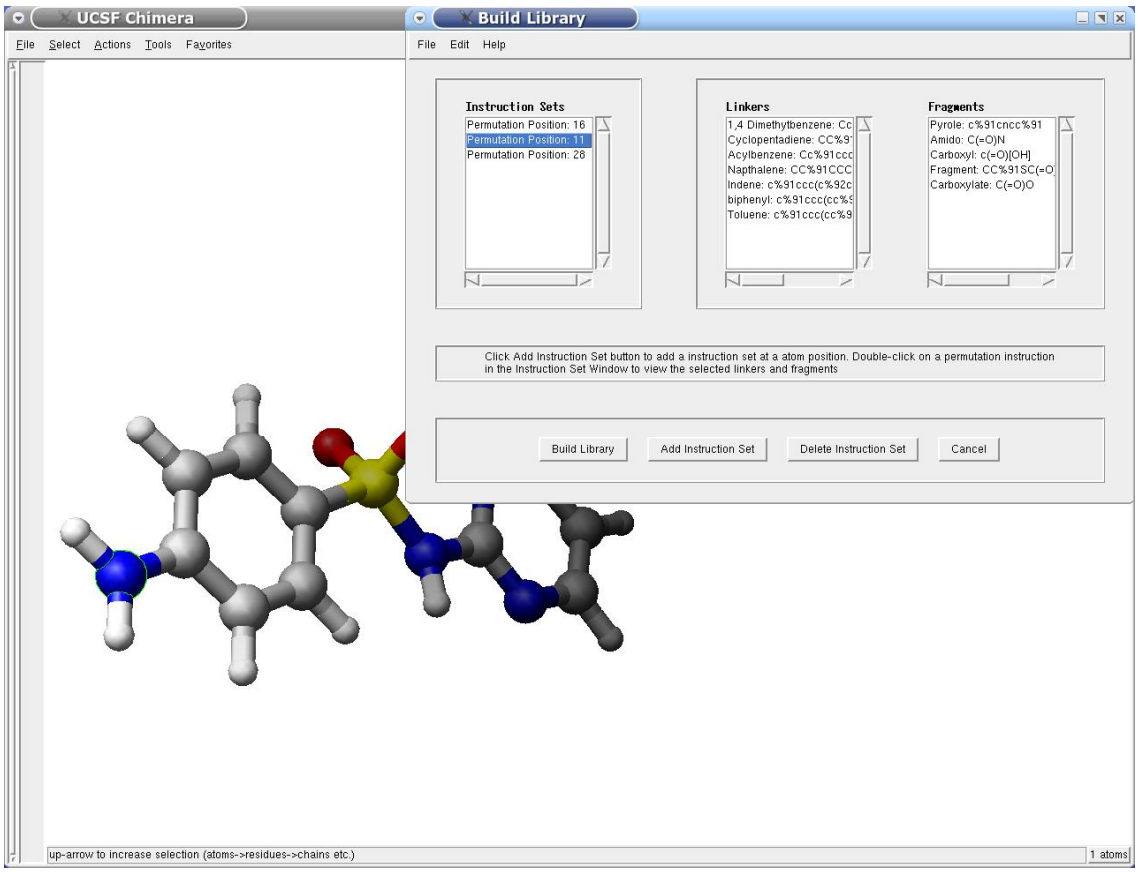



Figure 2.4: A screenshot of the LIGLIB Build Library window in Chimera. The instructions set listbox shows all the permutation positions already selected by the user. The Linkers and Fragments listboxes display all the linkers and fragments that were selected for the permutation position selected in the instruction set listbox.

The first input from the user is the scaffold in mol2 format, and can be loaded into Chimera before or after LIGLIB is run. The user would typically create a scaffold by taking a hit molecule relevant to the target being investigated, and removing fragments at positions where permutations over other fragments are to be done. LIGLIB allows the user to enter a

scaffold with explicit, partial or no hydrogens attached to it. However, the user will not be able to select hydrogens as permutation positions if they are not visible in Chimera. The scaffold is loaded into Chimera with the usual Open File dialog and is displayed in the Chimera visualization frame. LIGLIB does not access the input scaffold through the Chimera instance of it, but rather creates its own instance of the structure from the structure file. LIGLIB records the scaffold file path from the opened scaffold's `chimera.Molecule.openedAs` object, which returns a list of which the first entry is a string indicating the file name and directory path.

LIGLIB takes advantage of the graphical capabilities of Chimera when recording the selected permutation positions. When the user indicates that a new set of permutation instructions should be created, the user is prompted by LIGLIB to select a permutation position on the scaffold, which is done by holding down the ctrl-key and clicking on an atom. The atom selected by the user will be replaced by the first atom of the linker or fragment that is added to the scaffold during the molecular permutations. If a non-hydrogen atom was selected, it will be removed with all its attached hydrogens, if however a hydrogen was selected, only that hydrogen is removed. The atom in the fragment/linker that replaces the atom will be connected to all the atoms that were connected to the original atom on the scaffold.

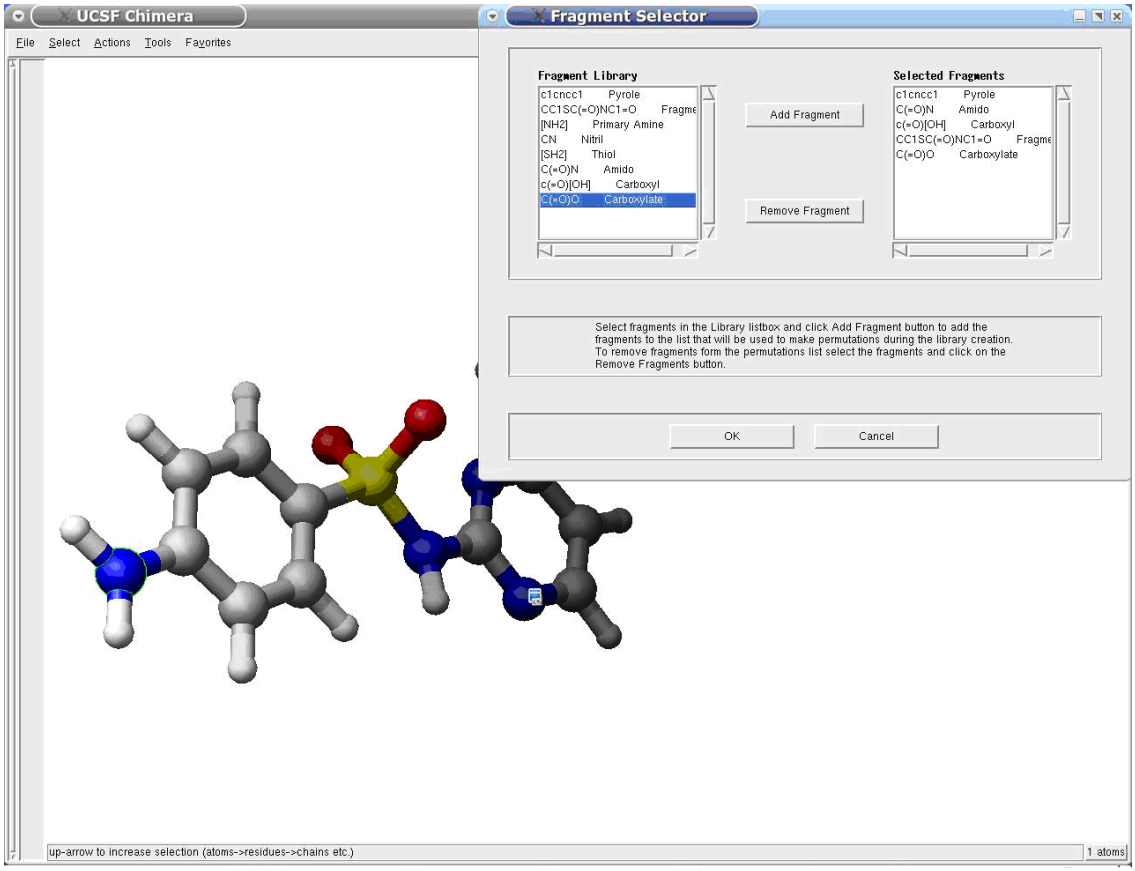



Figure 2.5: A screenshot of the Fragment Selector window in Chimera. The user selects fragments from the fragments library to make permutations during library generation.

After selecting a permutation position LIGLIB prompts the user with a Yes/No dialog that allows the user to indicate whether there will be linkers and fragments activated at the permutation site or just fragments. If the user selects to activate both linkers and fragments they are prompted with an open file dialog for each, which is used to open a linker and fragment library file respectively. Linkers and fragments are selected in the same manner. After the fragment or linker file is opened, a selector dialog appears which allows the user to add and remove fragments/linkers from the library file to and from the selected permutations group (Figure 2.5).

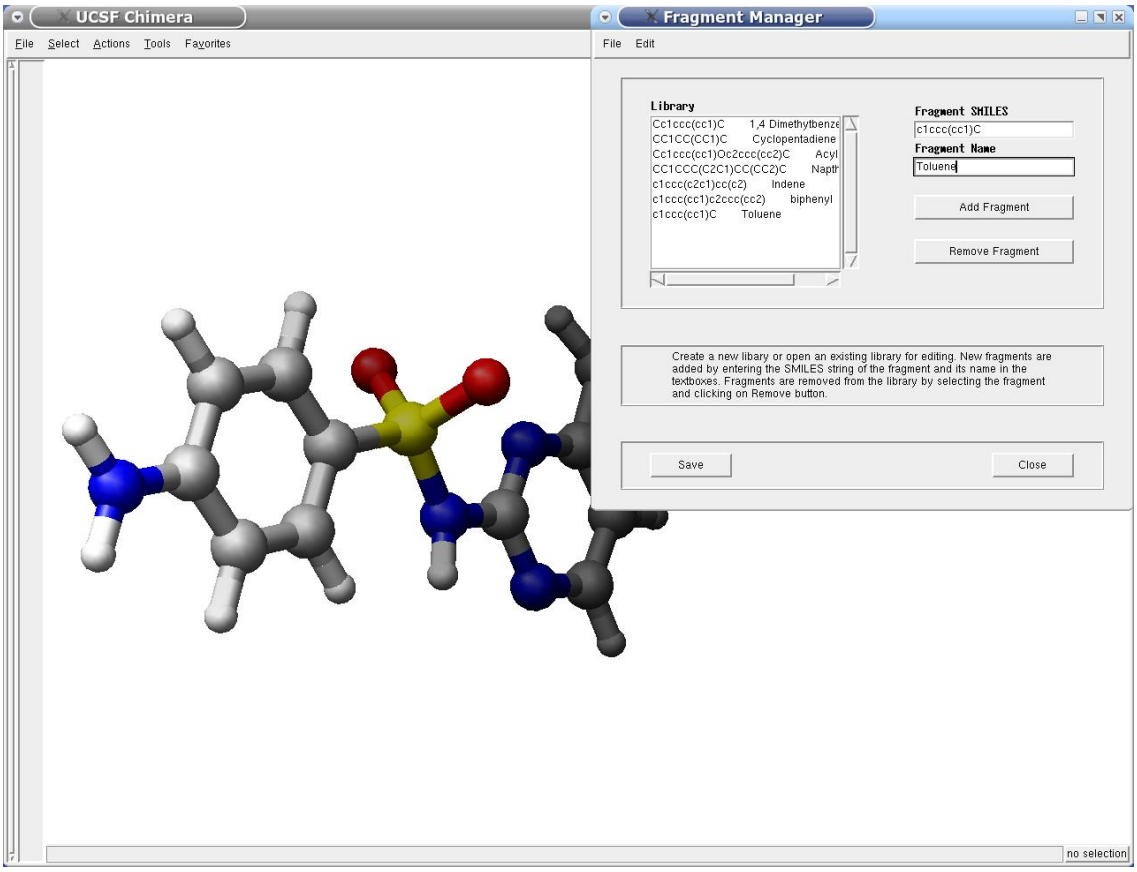



Figure 2.6: A screenshot of the LIGLIB Fragment Manager in Chimera. The user can create new or manage existing fragment/linker libraries. New fragments/linkers can be added and removed form the library using the manager.

LIGLIB allows the user to create new and manage existing fragment/linker library files with a fragment/linker manager tool (Figure 2.6). The user can add fragments/linkers to the library by entering a fragment/linker SMILES string and a description of it in the input textboxes, and remove fragments/linkers by selecting them in the library listbox and clicking the remove button. The order of the atoms in the SMILES string entered by the user will determine the attachment points on the fragments and linkers. For linkers, the first atom in the string will replace the atom selected on the scaffold structure while the last atom will attach the other end of the linker to a fragment (Table 2.3). Note in the last row of Table 2.4 that the last atom in the string is not used as the attachment point. When the string ends with a branch, the final atom in the branch is not used as a connection point, but rather the atom to which the branch is connected. If no linker set is used, the first atom in the SMILES string of a fragment will replace the atom selected by the user on the scaffold, otherwise it will attach to the end of the linker fragment (Table 2.4).

Table 2.3: Examples of fragments SMILES strings and their structure. Stars and underlines indicate attachment points.

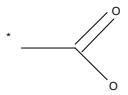
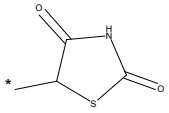
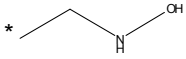
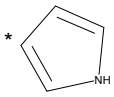
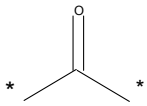
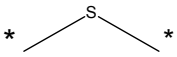
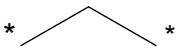
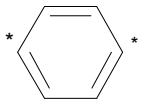
| SMILES String | Fragment |
|---|---|
| C(=O)O |  |
| CC3SC(=O)NC3=O |  |
| CCNO |  |
| c1cncc1 |  |

Table 2.4: Examples of linker SMILES strings and their structure. Stars indicate attachment points.

| SMILES String | Linker |
|---|---|
| CC(=O)C |  |
| CSC |  |
| CCC |  |
| c1ccc(cc1) |  |

### 2.2.2 Convert Scaffold into SMILES

Once all the input is gathered from the user and the Build Library button is clicked, LIGLIB starts the computational process. LIGLIB builds the library with the scaffold, linkers and fragments in SMILES format, which means that the input scaffold, a mol2 3D structure, should be converted to 2D SMILES format. It was decided not to write the code for the 3D to 2D conversion since an open source tool – OpenBabel – already exists which can do the conversion (Figure 2.7). OpenBabel is a program and library designed to translate the huge

number of chemical files and formats used by scientist. LIGLIB interfaces with OpenBabel by passing command-line commands to OpenBabel. The command format is `Babel -imol2 <infile> -osmi <outfile> -h` where `<infile>` is the file name and path of the input scaffold obtained during user input, and `<outfile>` is the scaffold file name with the extension ".smi" and will be the SMILES output file generated by OpenBabel. During the format transformation OpenBabel adds explicit hydrogens to the SMILES structure.

[CH3][C@H]1[C@@H]2[C@]34[C@H]([CH2][CH2]1)[C@@H]([C](=[O])[O][C@@H]3[O][C@]([CH2][CH2]2)([O][O]4)[CH3])[CH3]

Figure 2.7: The SMILES string of artemisinin created by OpenBabel from the 3D-structure mol2 file.

For LIGLIB to generate the library it needs to know where the permutations should be made in the SMILES structure, but the user indicated which positions to use on the 3D mol2 structure and there is no direct relation between the atom positions in the SMILES string and the positions in the mol2 file. It was therefore necessary to create a bijection algorithm that could map atoms in the mol2 file to their own instance in the SMILES string. The algorithm implemented was graph isomorphism-based. Both the 3D structure and the 2D SMILES structure can be represented as graphs with the atoms the vertices and the bonds between them the edges. Two graphs created from the 3D and 2D representations respectively of the same chemical structure will always be isomorphic to each other, which means they will both have corresponding vertices with the equal connectivity between them. The bijection algorithm exploits the identical connectivity attribute of the isomorphic graphs to find matching atom positions in the 3D and 2D structures. For each atom in a chemical graph a unique tree data structure can be created with the current atom as root of the tree, and children nodes the atoms connected with covalent bonds to a parent node atom. Two identical nodes in two isomorphic graphs can be mapped to each other if the connectivity trees drawn with both of them as the root are exactly the same.

For the bijection algorithm to work both graphs have to be isomorphic and LIGLIB has to ensure that this is the case when converting the 3D structure to the 2D structure and when creating the tree data structure. When OpenBabel transforms a 3D structure into a 2D SMILES structure it does so in either of two modes: with explicit hydrogens or without

hydrogens. The input scaffold can have no hydrogens, explicit hydrogens or just partial hydrogens attached to it. If the hydrogen states of the 3D and 2D structures are not the same, the graphs created from them will not be isomorphic. To address the isomorphic issue LIGLIB makes use of the add hydrogens functionality of OpenBabel. Explicit hydrogens are added to the SMILES string when it is created, and to the 3D structure before its representative graph is created. The 2D and 3D graphs will always be isomorphic since the same OpenBabel algorithm is used to add the explicit hydrogens for both the 2D SMILES and the 3D mol2 structure.

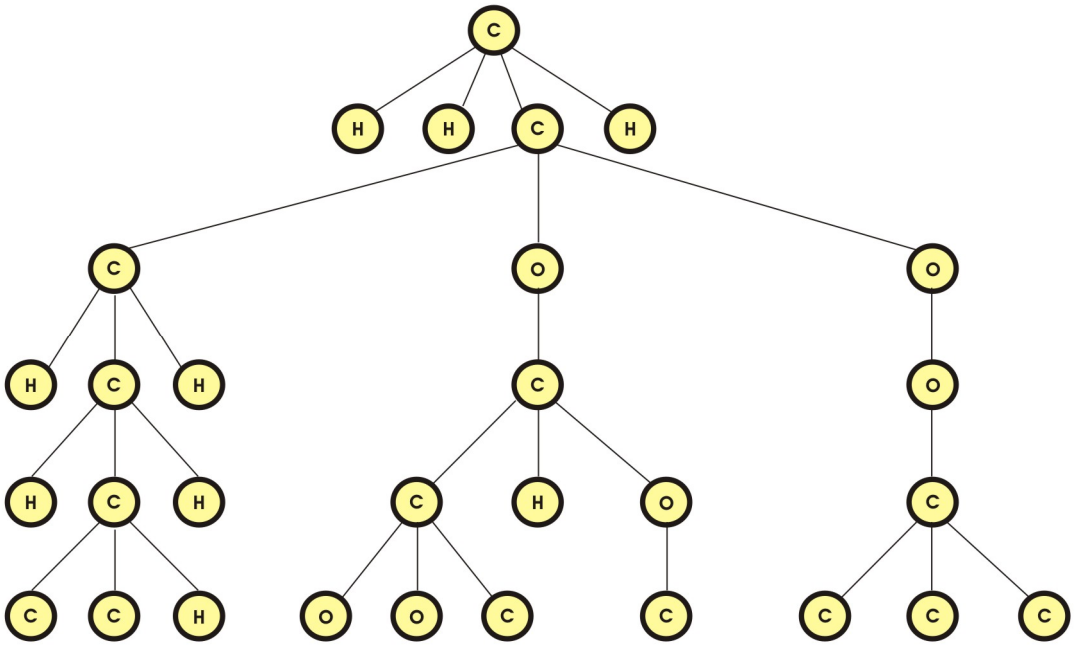



Figure 2.8: An example of a tree data structure up to depth level 6. The root is the atom selected by the user.

The bijection algorithm starts by creating a graph for the 3D and 2D structures. The graphs are represented as 2D arrays where each row represents a vertex and the columns references to the vertices that are connected to the current vertex, by edges. The 3D structure's graph is generated by building vertices for each atom in the mol2 file, and adding edges according to the bond information in the Bond attribute section of the file. The 2D structure's graph is generated by making vertices for each atom in the SMILES string, and adding edges according to the neighbors of the atoms in the string. The bijection algorithm then generates a tree data structure representative of the connectivity of the graph created from the 3D mol2 structure, with the atom position selected by the user used as the root of the

tree, followed by the recursive adding of nodes to the tree as the algorithm traverses the 3D structure's graph (Figure 2.8). The tree is built up from node objects with each node assigned an atom type. The connectivity between the nodes of the tree is recorded by pointers from each parent node to its child nodes. Creating a tree data structure for a chemical with cyclic structures will result in an eternal loop if the adding of nodes is not explicitly stopped. The bijection algorithm prevents the eternal loop situation by setting a maximum tree depth while creating the tree data structure, which means that tree building will stop once the tree reaches a certain depth, and for non-cyclic structures when the maximum depth level is reached or if the algorithm traversed the 3D graph up to all the terminal nodes.

The next step in the bijection algorithm is to find an atom position in the 2D graph that, if it is used as a root node, produces a tree data structure identical to the tree drawn for the atom position in the 3D graph. Instead of building a tree for each node in the 2D graph and comparing all of them to the 3D tree structure, the algorithm uses some heuristics to substantially reduce the search space. The first obvious heuristic is to investigate only 2D trees with the same atom type root as the tree created from the 3D graph. The second heuristic involves the building of the 2D graph trees. Instead of building 2D trees, a trace approach is used. The algorithm does not create a 2D tree but rather tries to traverse the 2D structure's graph with the 3D tree structure as a map (Figure 2.9). If the algorithm is able to successfully traverse the 2D structure's graph correctly using the 3D structure as a map, it follows that the vertex used as starting point for the traversal is the same vertex as the vertex in the 3D structure's graph used, which was used as root of the 3D structure's tree.

The algorithm recursively searches the 3D tree depth-first. At each recursion step two main tests are performed and true or false is returned to the parent node depending on whether both tests are passed. The first is whether the neighborhood of the tree node and the 2D graph node being compared are the same, and secondly, whether the children nodes of the current node return true indicating that they all matched. The first test needs to be true if the recursion is to continue with the children nodes, otherwise no further searching will be done for the children and the current node will return false to its parent. If the first test is true and the children nodes are recursively investigated and they all returned true, then the current node will also return true to its parent caller node.

```
create 2D_Graph and 3D_Graph
create 3D_Tree from 3D_Graph with user selected atom as root
FOR all 2D_Graph_Atom_Nodes
   IF 2D_Graph_Atom_Node match atom type of 3D_Tree_Atom_Root THEN
      IF Investigate_Neighborhood(2D_Graph_Atom_Node, 3D_Tree_Atom_Root)
      - True THEN
         record match between 3D_Graph_Atom_Root and 2D_Graph_Atom_Node
         break loop through 2D_Graph_Atom_Nodes
      ENDIF
   ENDIF
ENDFOR

Investigate_Neigborhood(2D_Graph_Atom_Node, 3D_Tree_Atom_Node)
   IF 3D_Tree_Atom_Node has no children nodes THEN
      return True
   ENDIF
   IF 2D_Graph_Atom_Node's and 3D_Tree_Atom_Node's children nodes types match THEN
      FOR all 3D_Tree_Atom_Child nodes of 3D_Tree_Atom_Node
         FOR all 2D_Graph_Atom_Child nodes of 2D_Graph_Atom_Node
            IF 3D_Tree_Atom_Child and 2D_Graph_Atom_Child match THEN
               Investigate_Neighborhood(2D_Graph_Atom_Child, 3D_Tree_Atom_Child)
            ENDIF
         ENDFOR
      ENDFOR
   ENDIF
   IF all 3D_Tree_Atom_Nodes had a Investigate_Neighborhood() call return True THEN
       return True
   ELSE
      return False
   ENDIF
```

Figure 2.9: Pseudocode of the of the bijection algorithm.

The 2D graph vertex matched by the bijection algorithm to the 3D graph vertex represents the user-selected atom in the SMILES string. LIGLIB finds the atom represented by the 2D graph vertex and removes it from the SMILES string. Non-hydrogen atoms are enclosed in square brackets with its attached hydrogens and if the user-selected atom is non-hydrogen the entire bracket set is removed so that the atom and its attached hydrogens are removed. If the selected atom is a hydrogen, LIGLIB finds the brackets set of the atom to which the hydrogen is bonded to and removes a hydrogen from it. If an entire bracket set was removed it is replaced by a unique marker. However, if a hydrogen was removed, the unique

marker will be placed directly after the bracket set that contained the hydrogen. The markers indicate where molecular permutations should be made during the library-building step. The markers are sequential number characters starting from zero, with a pre- and post-underscore and are distinguishable from the regular SMILES format.

### 2.2.3 Generate the Chemical Library

The permutation instructions are made up of Permutation Instruction Sets, one for each permutation position, and for each Permutation Instruction Set there is a range of linkers and fragments. If LIGLIB permutes over all the linkers and fragments at each of the positions, then the total number of new molecules generated in the library will be the result of equation (2.1). Each bracket set in the equation represents a single permutation instruction set at a single permutation position and is made up of the number of linkers ($l_x$) and fragments ($f_x$) that will be used to make permutations on the structure. The number of bracket sets is equal to the number of permutation instruction sets, which depends on how many permutation points are used. The number of linkers and fragments does not have to be the same for each of the permutation instruction sets. Linkers are optional and for permutation instruction sets with no linkers the $l_x$ variable will be left out of the equation and the bracket set will then be $(f_x)$.

$$F(l, f) = (l_1 \times f_1) \times (l_2 \times f_2) \times (l_3 \times f_3) \times (l_n \times f_n) \quad (2.1)$$

When the library is generated all the possible combinations of the selected fragments and linkers at each of the permutation positions are permuted over and for each permutation combination a new molecule is created and added to the library. The library algorithm is explained by example using simple linkers and fragments from tables 2.3 and 2.4. There is no experimental relevance to the example. The algorithm performs the permutations by running through two recursions. The first recursion traverses a 2D array made up of all the fragments activated for each of the permutation positions (Figure 2.10). The algorithm starts a recursion at each of the fragments in the top row of the array. The recursion runs until it reaches the bottom row of the array, and then starts the second recursion. In this fashion the recursion runs through all the possible paths from the top to the bottom of the array and starts a secondary recursion each time it reaches a bottom array entry. For each of the secondary

recursions that are started a new array is created and traversed (Figure 2.11). The secondary array is created by taking the path followed by the primary array, and combining the fragments at each step with all the linkers activated at that permutation position. The secondary array rows represent the permutation positions and the columns the range of linkers attached to the fragment being used as a step in the primary recursion. The secondary recursion is run in the same manner as the primary one, finding all the possible paths to the bottom of the secondary array. Each time the recursion reaches the bottom of the array a new molecule is created using the path followed to the bottom. Each fragment-plus-linker used as a step in the recursion path is attached to the scaffold structure at the respective permutation points.

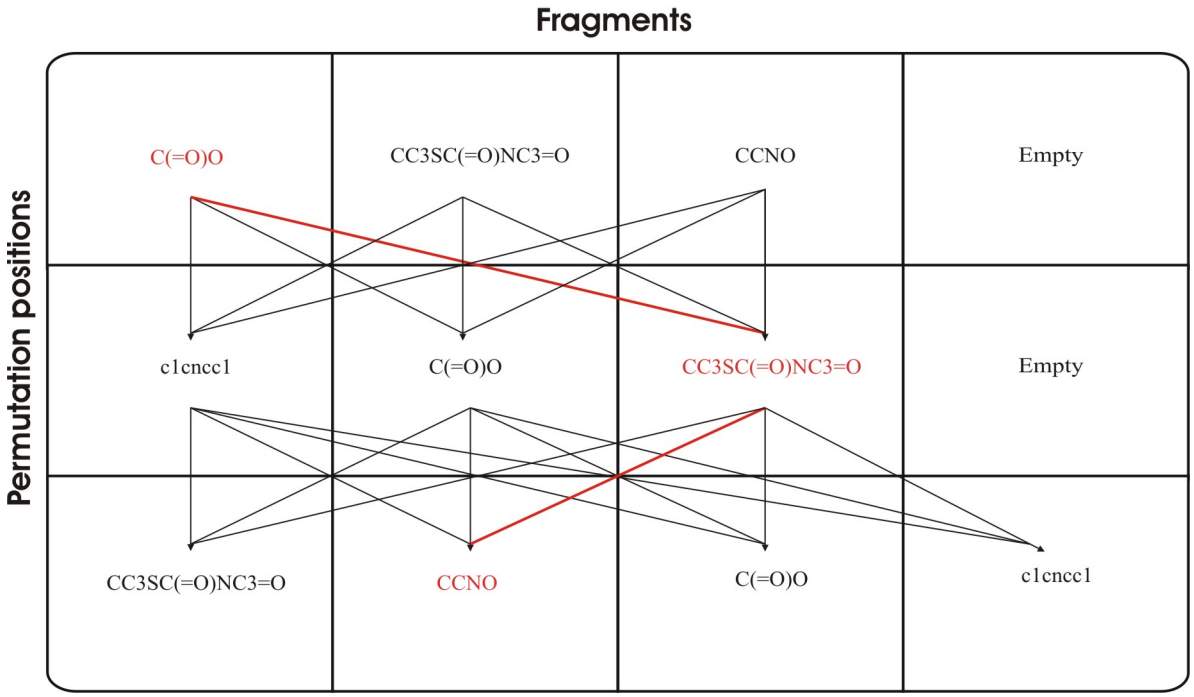



Figure 2.10: A diagram illustrating the primary recursion of the library building algorithm. Each row represents a permutation position and each block contains a fragment selected by the user. An example recursion path is indicated in red.

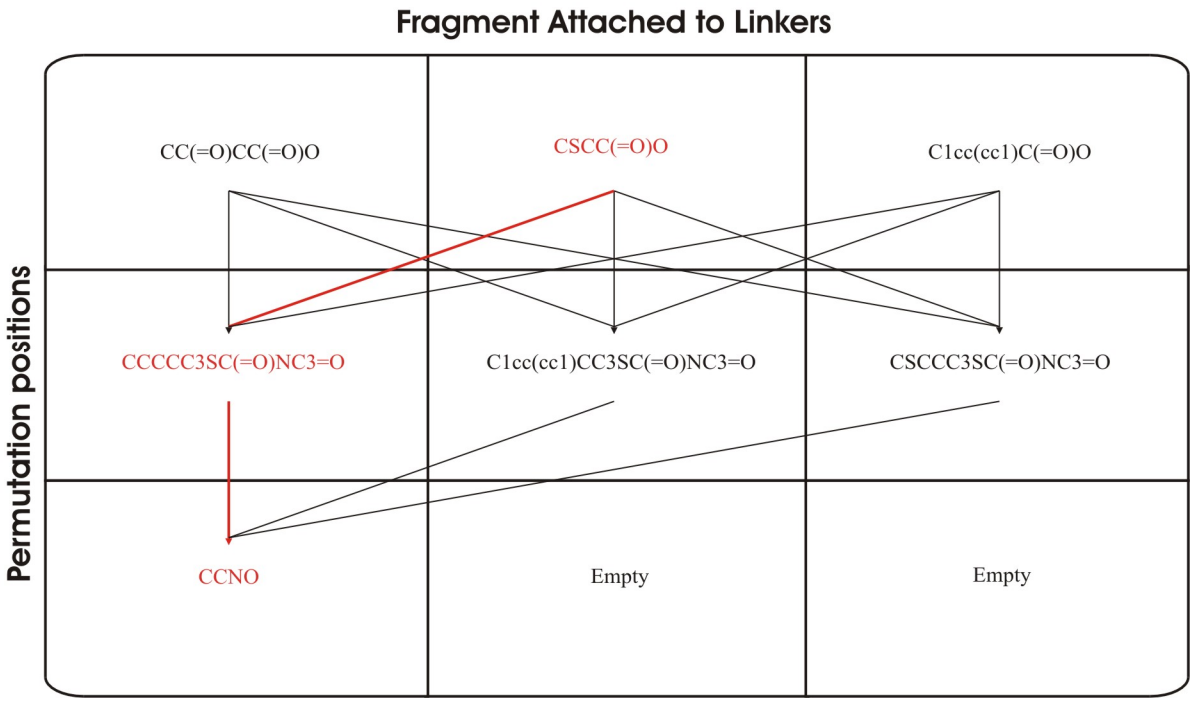



Figure 2.11: A diagram illustrating the secondary recursion performed by the library-building algorithm. Each row represents a permutation instruction and each block contains a composite of a fragment and a linker. The composite fragments are inserted into the scaffold SMILES string during library-building. An example recursion path is indicated in red.

The secondary recursion paths are made up of a number of steps equal to the number of permutation positions and each step in a specific path is a SMILES composite of a linker and a fragment. The Composite fragments are attached to the scaffold structure by inserting it into the scaffold SMILES string at the molecular positions selected by the user. Each permutation position is marked in the SMILES string by a unique marker inserted during the scaffold to SMILES step. The marker is removed and replaced by the composite fragment used as a step in the secondary recursion. To ensure that the connectivity of the SMILES scaffold remains the same the composite fragment is inserted into the SMILES string as a branch structure and is enclosed by branch brackets before insertion. An unbranched insertion will bond the first atom of the composite fragment to the atom before the user selected one, and the final atom if the composite fragment to the atom following the one selected by the user, which results in an fragment insertion in to the scaffold instead of the intended fragment attachment to the scaffold. SMILES format uses digits in the string to indicate ring closures. The same digit found consecutively in the string indicates the opening and closing positions of a ring. If a composite fragment is inserted in the scaffold in-between a ring's open and closing digits and it has uses the same ring closure digit as the one in the scaffold, then SMILES interpreters

will use the ring open digit from the scaffold and the ring open from the composite fragment to form the one ring, and the closure digits of the scaffold and composite fragments as the other ring. To prevent this incorrect formation of rings the composite fragment is not allowed to use the same closure digits as that used by the scaffold. LIGLIB replaces the ring closure digits in the composite fragment with closure markers in the nineties. Double-digit closures can be used in SMILES by adding a "%" character directly in front of it. The new closure range used by LIGLIB will not clash with the ring closures used for the scaffold since OpenBabel reuses ring closure digits when it converts the scaffold to SMILES format and it will never need to have more than ninety open rings to represent a molecule in SMILES.

```
create Position_vs_Fragment[][]
FOR each Fragment in Position_vs_Fragment[Row = 0][]
   Fragments_Recursion(Row = 0, Column = Fragment)
ENDFOR

Fragments_Recursion(Currnet_Row, Current_Column)
   IF Recursion at final PermutationPosition THEN
      create Position_vs_LinkerPlusFragment[][]
      FOR each LinkerPlusFragment in Position_vs_LinkerPlusFragment[Row = 0][]
         Fragments_Plus_Linkers_Recursion(Row = 0, Column = LinkerPlusFragment)
      ENDFOR
   ELSE
      FOR each Column in Position_vs_Fragment[Current_Row]
         Fragment_Recursion(Row + 1, Column)
      ENDFOR
   ENDIF

Fragments_Plus_Linkers_Recursion(Current_Row, Current_Linker)
   IF Recursion at final PermutationPosition THEN
         Save scaffold with attached LinkersPlusFragments according to recursion
         path
   ELSE
      FOR each Column in Position_vs_LinkerPlusFragment[Current_Row]
         Fragments_Plus_Linkers_Recursion(Row + 1, Column)
      ENDFOR
   ENDIF
```

Figure 2.12: Pseudocode for the Library building algorithm. Two recursion functions are used.

### 2.2.4 Convert Library into 3D Structures

During input the user was prompted to select a directory where the library should be created. In that user selected directory two folders are created – SMILES_Library and Mol2_Library – and a version of the ligand library is stored in each. The 2D output library is simply the SMILES molecules created by LIGLIB in the Build Library step. The 3D library however has to be generated from the 2D library. The generation of 3D coordinates is a complex problem and at the time of writing this document only two tools were available that could successfully perform the coordinate generation, Corina and Concord, of which Corina was selected for this study. Corina has been under development since 1984 and the recent version has proven to generate good 3D coordinates (Schönberger *et al.,* 2000). Corina is a proprietary package and cannot be packaged with LIGLIB. The user will be required to have it installed locally on the workstation if LIGLIB is to be used to generate the mol2 library. If Corina is not locally installed LIGLIB will give only the SMILES library as output. LIGLIB interfaces with Corina through command line commands in the form `corina.lnx -I t=smiles -o t=mol2 <infile> <outfile>` where `<infile>` is the SMILES file used as input and `<outfile>` is the mol2 output filename.

## 2.3 Discussion

The methodology described in this chapter was successfully implemented in LIGLIB and allows a user to build a library of analogous ligand structures by taking a scaffold, selecting permutation positions on it, and selecting linkers and fragments that are to be used during the permutation of the structure. The method allows for fast generation of ligands that are analogous of a molecule of interest. The user guides the library building by selecting permutation positions close to interaction points in the active site and selecting fragments that could possibly take part in interactions. Linkers can be used to vary the distance between the fragment and the interaction site in the protein, or can even take part in interactions themselves.

SMILES is an easy format to work with small molecules and is very popular for working with small molecule libraries and databases. If the permutations were made in 3D, molecular and quantum mechanics would have had to be incorporated to build the new molecules in correct conformations. Incorporating molecular and quantum mechanics would have been

extremely difficult and would have had a detrimental effect on the computational time that LIGLIB would have taken to build the library, and LIGLIB was designed to be a fast easy tool to create ligand libraries. The user would also have had to create or find 3D fragments and linkers to use for the molecular permutations and such libraries are not readily available. SMILES fragments on the other hand are very easy to create and the user can create any new fragment or linker by simply typing a string according to the SMILES conventions. LIGLIB has a fragment/linker library manager tool that the user can use to build new and manage existing libraries. The libraries are saved in files and can be reused for multiple permutation sites in a single ligand library building run, and for other ligand library build runs. Generating molecules in SMILES format is computationally inexpensive and a large amount of new molecules can be generated in a small amount of time.

During the design of LIGLIB the 2D to 3D coordinate transformation was considered. It is, however, a very difficult problem and the most well known two stand-alone applications that are able to properly generate coordinates from SMILES – Corina and Concord – have been developed over more than 20 years, which illustrates the complicated nature of the problem. It was therefore decided not to attempt the 3D coordinate generation and rather use an existing tool. There is no recent publication directly comparing Corina and Concord and both companies distributing the tools claim that their tool produces the best results. However, personal communications with persons in the cheminformatics field suggested that Corina produced better results and there is a fair amount of publications where Corina was used successfully (Lapinsh *et al.,* 2005; Singh *et al.,* 2006; Song *et al.,* 2005). Based on this and in the absence of a scientific comparison, it was decided to use Corina for the 3D coordinate generation. Both Corina and Concord are proprietary applications, which means that they cannot be redistributed with LIGLIB. Therefore, if the user would like to have a 3D mol2 library as output with the 2D SMILES one, Corina has to be installed locally on the workstation. LIGLIB interfaces with Corina and uses it to create the 3D library.

SMILES notation is able to capture the chirality of molecules, and OpenBabel does make use of the SMILES chirality notation during the file conversion. Corina then use this chirality information to generate the correct structure conformations in the output library.

The tree algorithm implemented has been shown to work correctly to perform the graph bijection between the SMILES and mol2 structures. Though it should be mentioned that the algorithm is only suitable for ligands, and will not scale well to larger molecules like proteins.

Also an alternative method could have been used. If a new mol2 to SMILES format parser was developed for LIGLIB it would have been possible to map the bijection between atom in the SMILES and mol2 structures as the SMILES was created. However, OpenBabel is a proven molecular format parser and it was decided to rather use it as the SMILES parser of LIGLIB and to develop an isomorphism to perform the bijection.

# Chapter 3

# Development, Architecture and Design

## 3.1 Development

### 3.1.1 Development Process

No iterative development process could be found to fit as development model for LIGLIB. LIGLIB has only one major user interaction function – to build ligand libraries – and it cannot be subdivided into further user-view logical functionalities. Also, virtually all the LIGLIB code has to run to perform the build ligand library function. For an iterative process, there has to be a set of functionalities to iterate over and develop sequentially. Since there were no logical subsets of functionality in LIGLIB and it was decided to follow the older, yet proven waterfall development process. Even though the waterfall model has been used successfully for many object-oriented projects, it has lately been considered outdated in comparison with the newer, more popular iterative processes and has a high level of risk associated with it. However, since LIGLIB does not fit one of the iterative models, it was decided to follow the waterfall process – not dependent on user-view functional subdivision – which would fit the LIGLIB project sufficiently as long as the risks were managed.

### 3.1.2 LIGLIB's Python Front-End

LIGLIB was developed with a Python front-end and a C++ back-end. The front-end is responsible for user interactions, gathering of the input information and, interacting with the vector application by making use of its functionality.

The decision to use a Python front-end was based one three major considerations: Firstly, the Chimera objects are available as Python code and are easily accessible to the Python

LIGLIB front-end, which makes extension building simpler. Secondly, Chimera has a version of Python (v2.3) and the IDLE Integrated Development Environment (IDE) imbedded in it to create an extension-building environment where the user can write extension Python code in IDLE while the Chimera Python version interprets and imbeds it into Chimera. Thirdly, the Chimera GUI was developed with the Tkinter Python interface to the Tcl/Tk GUI toolkit and it made sense to build the LIGLIB GUI with the same Python tool since it would avoid new software dependencies.

Chimera extensions are imbedded by coding the extension in a Python package and registering it with the Chimera Extension Manager, which keeps track of extensions. In the extension Python package folder, Chimera recognizes a special file named "ChimeraExtension.py" that contains a class derived form the Chimera class "chimera.extension.EMO", and which is built according to the Chimera extension building conventions. The derived class contains functions that return extension specific attributes indicating how the extension is embedded in the Chimera GUI, as well as an `activate()` function that initializes and runs the functionality that should be invoked in the package when the user starts the extension from the Chimera menu.

### 3.1.3 LIGLIB's C++ Back-End

The C++ back-end of LIGLIB contains the code responsible for building the ligand library. One of the main requirements of LIGLIB was that it should execute quickly, and even though it is primarily used to build small focused libraries, it was also required that it is able to building large ligand libraries. To meet these requirements it was decided to develop the computationally intensive code in C++, which is a compiled language and executes much faster than Python, an interpreted language. Developing in C++ is much more challenging than the newer programming languages and the standard C++ libraries do not support commonly-accepted functionality such as garbage collection. However, difficulties like these can be overcome by enforcing proper programming methodologies. The back-end was developed in the Eclipse IDE (Rivières & Wiegand, 2004) with the C/C++ Development Tools (CDT) plug-in. The C++ and Python bindings were created with the interface tool, SWIG (Simplified Wrapper and Interface Generator). SWIG (Cottom *et al.,* 2003) takes a SWIG-interface file with wrapping instructions as input, and generates a C++ and a Python source file. The generated C++ file and the LIGLIB back-end C++ files are compiled and the

object files linked into a shared library accessible to the Python wrapper file created by SWIG. The C++ back-end is now available to all Python programs that include the Python SWIG wrapper module. The LIGLIB back-end was developed in a Linux environment and was compiled with the GCC compiler.

### 3.1.4 Open Source Licence Agreement

Chimera is an open source application, and in the spirit of open source it was attempted at all levels of development, architecture and design to keep software extensibility in mind. The definite design abstraction of the LIGLIB functionality in the C++ back-end from the Python interface front-end means that all the LIGLIB functionalities are portable, and can be used to extend virtually all vector applications – e.g. VMD (Humphrey *et al.,* 1996), PyMol (http://www.pymol.org) and Jmol (http://www.jmol.org) – by simply creating a new front-end to access the back-end C++ code *via* the LIGLIB interface and the SWIG wrapper.

## 3.2 Architecture

The high level architecture of LIGLIB is made up of five main elements: Chimera vector application, LIGLIB's Python Front-end, LIGLIB's C++ Back-end, Corina coordinate generator and the OpenBabel format converter (Figure 3.1).

Chimera interacts with LIGLIB through a Python front-end, which is connected to Chimera with the Chimera Extension Manager tool. The Extension Manager tool extends Chimera with LIGLIB and makes is available from the tools dropdown menu. The Python front-end contains the GUI code and is responsible for all interactions between LIGLIB and the user. It also serves as layer between LIGLIB and Chimera, using access to the Chimera object model to call on functionality as required during the user input gathering in the Chimera environment. The input from the user is passed on to the LIGLIB C++ Back-end, which performs the computationally intensive task of building the ligand library. The Python front-end interacts with the C++ back-end with bindings created with the SWIG wrapping tool. SWIG created bindings allow the Python front-end to access a C++ back-end interface class as if the interface was a Python module. During the building process LIGLIB makes use of standalone applications when it converts the input scaffold into SMILES format, and the

output library into mol2 format. The SMILES conversion is done with OpenBabel and the mol2 conversion with Corina. The LIGLIB back-end interacts with both applications by passing command-line arguments describing input, action and output parameters. The command-line interface allows the user to work with any version of the Corina and OpenBabel as long as the command-line arguments do not change.

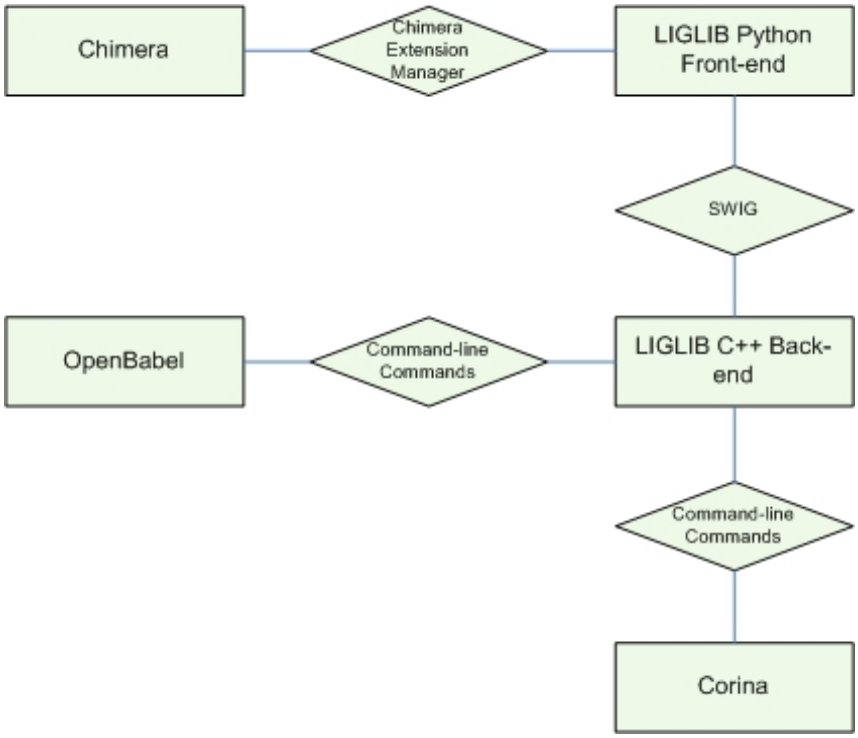



Figure 3.1: High level architecture of LIGLIB. The blocks are elements in the architecture and the diamonds the type of linkage between them.

## 3.3 Design and Implementation

### 3.3.1 Use Case View

LIGLIB has only two high-level functional interactions with the user, to build ligand libraries and to create and manage the linker and fragment libraries used during the library building process (Figure 3.2). The Manage Fragment or Linker Library use case (Figure 3.3) represents the process where the user creates and/or manages the libraries of linkers and

fragments that LIGLIB uses as molecular building blocks during the library build. The Build Ligand Library use case (Figure 3.4) represents the process during which the user gives permutation instructions to the system and initiates the library building.

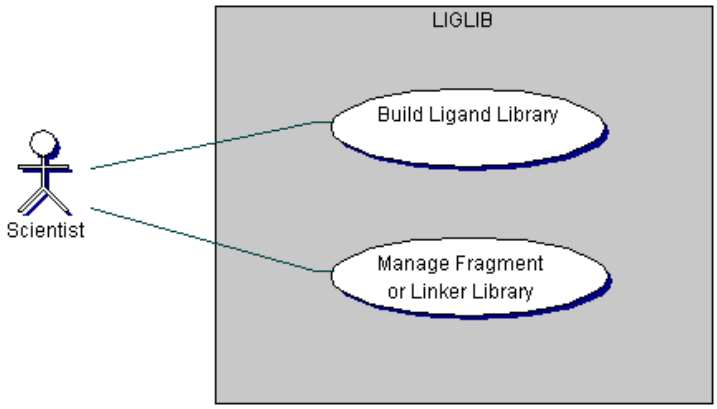



Figure 3.2: Use case diagram for LIGLIB.

**Manage Fragment or Linker Library**

Main Success Scenario:

1. Scientist runs LIGLIB from Chimera tools menu
2. Scientist selects to start the Fragment/Linker manager
3. Scientist selects to create a new library
4. Scientist adds and removes Fragments/Linkers to and from the library
5. System saves library

Extensions:

3a: Scientist selects to open an existing library

.1: System opens and displays existing library, returns to MMS at step 4

Figure 3.3: Textual use case for the Manage Fragment or Linker Library use case (Figure 3.2).

**Build a Ligand Library**

Main Success Scenario:
1. Scientist runs LIGLIB from Chimera tools menu
2. Scientist selects Build Ligand Library
3. Scientist loads a scaffold structure in Chimera
4. Scientist selects to add a Permutation Instruction
5. Scientist selects a Permutation Position on the scaffold
6. Scientist indicates whether Fragments, or Fragments and Linkers will be used
7. System opens a Fragment library selected by the scientist
8. The Scientist selects Fragments from the Fragment library
9. System records current Permutation Instruction
10. System builds ligand library

Extensions:
7a: Scientist indicated that Fragments and Linkers will be used
.1: System opens a Linker library selected by the scientist
.2: The Scientist selects Linkers from the Linker library, returns to MMS at step 7
10a: Add more Permutation Instructions
.1: Return to MMS at step 4

Figure 3.4: Textual use case for the Build Ligand Library use case (Figure 3.2).

### 3.3.2 Activity and Class View

LIGLIB is made up of 9 front-end and 12 back-end classes, excluding the classes created by SWIG (Figure 3.5). The Python front-end, embedded in Chimera, accesses the C++ back-end functional classes *via* the Type interface class, TLIGLIB. The full Chimera object model is made available to the front-end by importing the Python Chimera object module. The TLIGLIB class serves as the interface between the front-end and the back-end. The Python version of the TLIGLIB class is visible to the front-end and has the same functions as the C++ version. When one of the Python TLIGLIB functions is called by the front-end, it calls the true functions of the C++ TLIGLIB via the SWIG bindings.

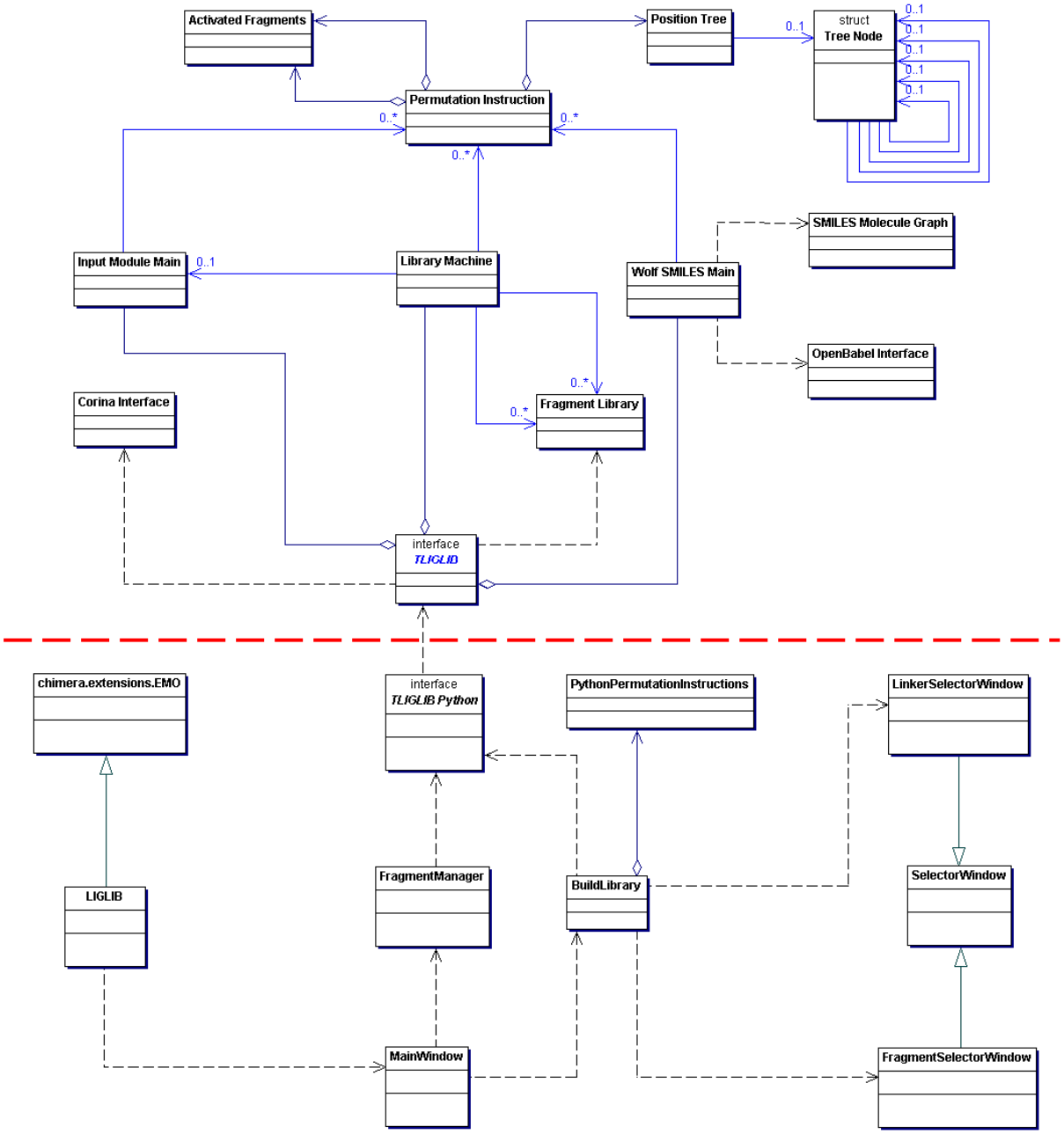



Figure 3.5: Full class diagram of LIGLIB. The red dashed line indicates the separation between the front-end and the back-end. Above the line is the back-end and below it the front-end.

The fragment/linker building blocks, used by LIGLIB to build the library, are stored as flat-files (Figure 3.6). The files are made up of tab-delimited strings in columns and rows. The rows are the fragment entries and the columns are respective entries for unique identifiers, SMILES fragments and fragment names. The user can create new libraries or open existing library files for editing. When editing or creating a new library, LIGLIB creates a Fragment_Library instance (Figure 3.7). The front-end FragmentManager object records library modification actions by the user and passes the new or edited fragments with their corresponding names to the Fragment_Library object via the TLIGLIB interface. When the user selects to save the library, the modified Fragment_Library is made persistent, and is saved to the original file, or a new file, depending on the user's instructions.

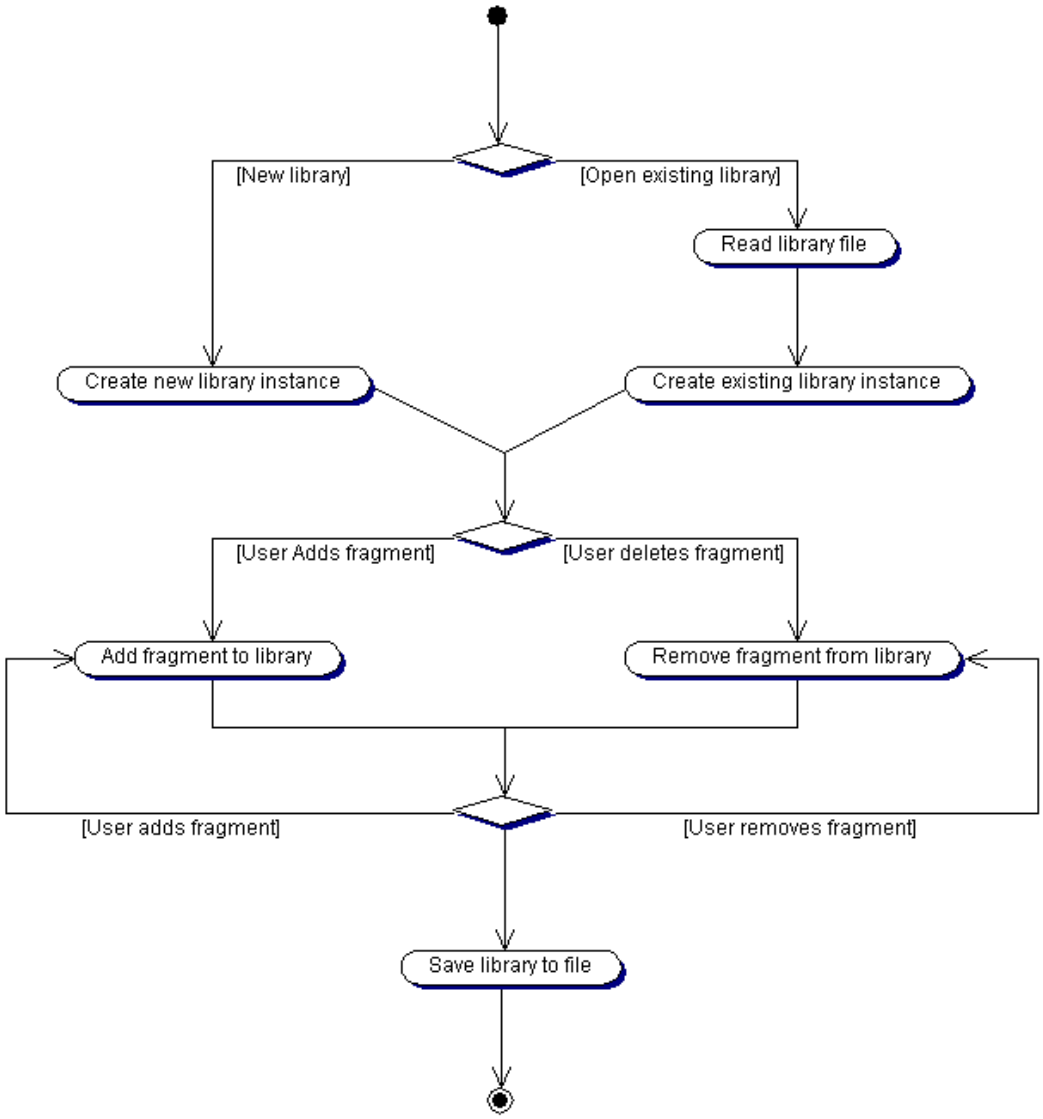



Figure 3.6: Activity diagram representing the management process of a Fragment/Linker library.

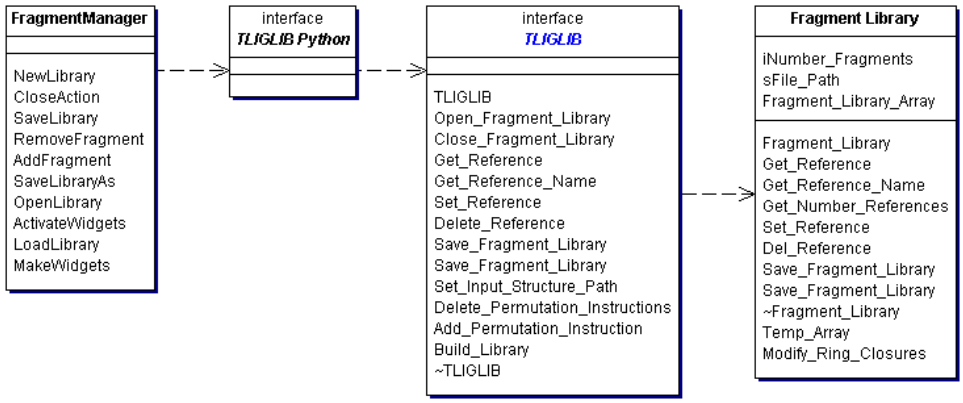



Figure 3.7: Classes involved in the management of Fragment/Linker libraries.

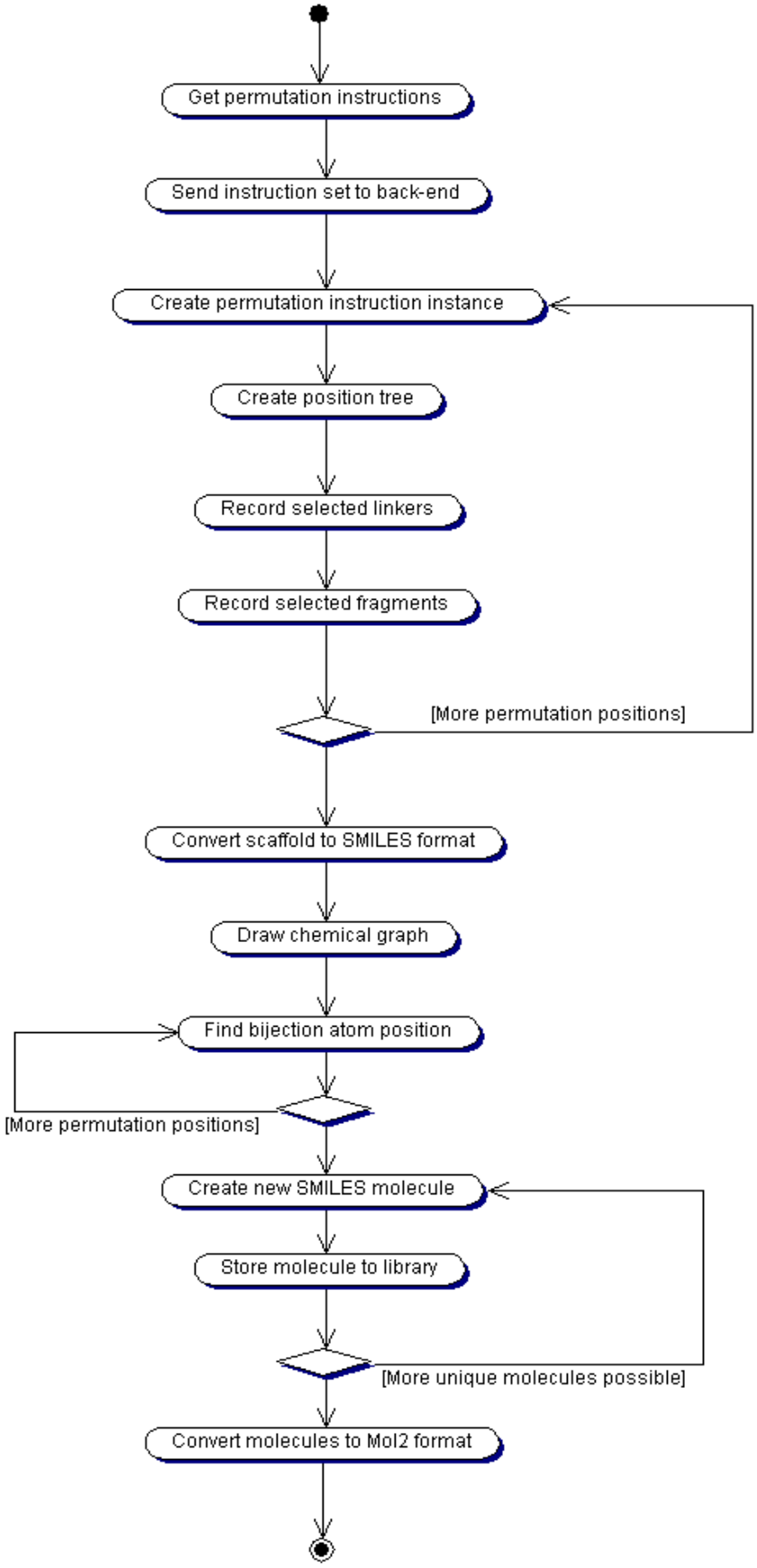



Figure 3.8 Activity diagram representing the library building process.

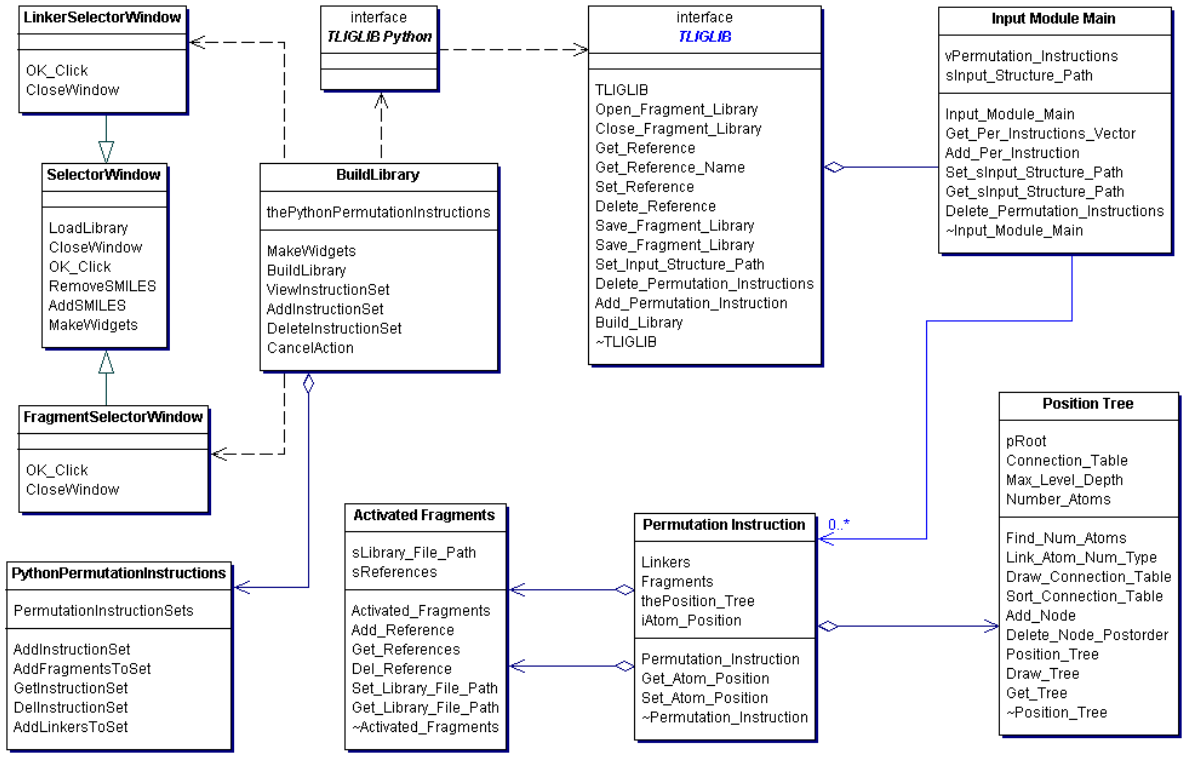



Figure 3.9: Classes involved in the input stage of execution.

The BuildLibrary class and its widgets in the front-end are responsible for managing the instruction input process and allowing the user to add and remove permutation instructions (Figure 3.9). Permutation instructions are recorded as instruction sets – one per permutation position – and LIGLIB gets instructions from the user one set at a time. The permutation instructions collected by the BuildLibrary class are recorded in the PythonPermutationInstructions class, which contains a list of which each entry is a permutation instruction set. Each permutation instruction set has an atom position, a list of linkers and a list of fragments. LIGLIB first allows the user to select a permutation position on the scaffold where permutations should be made. The selection is made in Chimera and LIGLIB has to get the selected atom position from the Chimera object model by calling chimera.selection.currentAtoms(), which returns a atom object from which the selected atom's mol2 atom ID is recorded and used as reference to the selected atom. The atom selection is tested to make sure that only one atom is selected and that it has not already been selected for another permutation instruction. Once the user has selected a permutation position, the LinkerSelectionWindow and FragmentSelectorWindow classes, – both of which are implementations of SelectorWindow – and their widgets record the linkers and fragments

to be used as building blocks at the current permutation position. The user selects a fragment/linker library file to use and the SelectorWindow classes access it via the Fragment_Library class from the back-end. For each fragment or linker selected from the library only its unique identifier is saved. If the user finished giving permutation instructions, the library build process can be started. The BuildLibrary class transfers all the user inputs from the front-end to the back-end class, Input_Module_Main, via the interface. Input_Module_Main has a reference to the input scaffold and a C++ vector containing permutation instruction sets. Each permutation instruction set is composed of two Activated_Fragments classes, for the linkers and fragments respectively, and a Position_Tree class. Each Activated_Fragment class contains a reference to a library file and a set of fragment reference IDs indicating which fragments/linkers were selected from the library as molecular building blocks. The Position_Tree class contains a set of functions and attributes to build a tree data structure, and a pointer to the root node of the tree, which is the atom selected by the user as the permutation position for the current permutation instruction set. Each node of the tree is represented by a struct object, which has an atom type attribute and a set of pointers pointing to leave node structs. The tree data structure represents all the possible molecular traversal paths of a molecule starting at the position selected by the user, and can identify all atoms in the molecule uniquely. Each node has a single parent node, which is the atom traversed before the current atom, and 5 possible leave nodes that are the possible paths that the traversal can take further. In the current version of LIGLIB the tree is given a maximum depth level, which prevents infinite loops from forming when a chemical graph containing circular structures is traversed.

The actual building of the library takes place in four main steps: Converting the scaffold structure into SMILES format, finding the permutation positions on the SMILES structure, making the permutations and generating the new library and finally, converting the library into 3D mol2 format (Figure 3.8 & 3.10).

The scaffold is converted to SMILES format using OpenBabel functionality accessed *via* the OpenBabel_Interface class. The Wolf_SMILES_Main passes the scaffold reference to the OpenBabel_Interface, which then passes it with execution parameters to OpenBabel. First, a command sent to OpenBabel instructs it to add explicit hydrogens to the scaffold 3D structure. A second command is sent and instructs OpenBabel to convert the scaffold to SMILES format. OpenBabel creates a file containing the SMILES scaffold, which is then read by the OpenBabel_Interface and returned to the Wolf_SMILES_Main class as a SMILES string.

The Wolf_SMILES_Main class is primarily responsible for finding the permutation positions in the SMILES string and inserting markers at those positions. The SMILES string with the markers inserted is referred to as WolfSMILES strings. A permutation site marker is inserted for each instance of Permutation_Instruction contained by Input_Module_Main. The SMILES permutation positions are determined with the Tree-trace algorithm (Section 2.2.2), which creates a graph of the SMILES molecule and traverses it with the Permutation_Instruction's Position_Tree as a map.

The library_Machine class is responsible for building the library. The library machine makes the permutations at the marker positions using the appropriate Permutation_Instruction's linkers and fragments. There is an Activated_Fragments instance for the linkers and the fragments respectively. Each Activated_Fragment instance contains a reference to a fragment/linker library file and the unique identifiers of the fragments/linkers selected by the user. The Library_Machine retrieves the selected fragments/linkers SMILES strings from the library files using an instance of Fragment_Library. The permutation algorithm (section 2.2.3) builds the new molecules by modifying the scaffold SMILES string at the permutation positions with the relevant SMILES fragment/linker SMILES. The SMILES output library is saved under a user-defined directory.

Upon completion of the library built, Corina is used to convert the 2D SMILES library into 3D mol2 format and to generate coordinates for the atoms. Corina is accessed by Corina_Interface which passes commands to Corina *via* the command line.

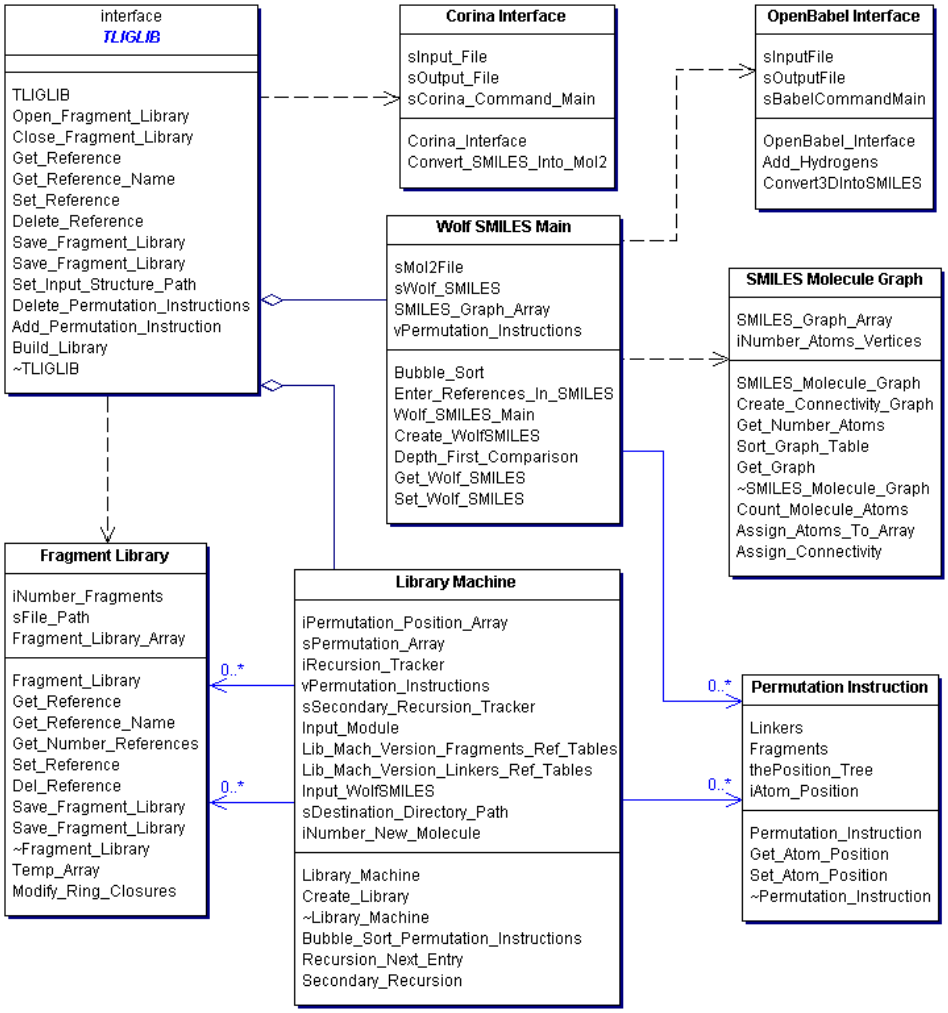



Figure 3.10: Classes responsible for building the ligand library.

## 3.4 Discussion

LIGLIB was designed while adhering to fundamental software methodologies and techniques. The resulting design is well abstracted from the complexities of the scientific problems encountered and will be easy to maintain and edit by other open source developers. The back-end portability will allow LIGLIB to be implemented in any visualization toolkit by simply writing code to interact with the back-end interface.

# Chapter 4

# Validation Study on *Plasmodium falciparum* Glutathione S-Transferase

## 4.1 Introduction

### 4.1.1 Aim

The goal was to build an *in silico* library of putative *Plasmodium falciparum* Glutathione S-transferase (*Pf*GST) inhibitors through a rational drug design implementation of LIGLIB in conjunction with molecular docking and screening, thereby validating LIGLIB.

Validating LIGLIB would show that it was implemented correctly and that the Markush molecular enumeration technique is a powerful approach to drug discovery. It also serves as an example of how LIGLIB can be applied in rational drug design.

It was hypothesized that building small fragments according to a key and lock model in localized portions of the active site, and then using these fragments as molecular building blocks in a process where all the possible combinations of these rationally designed smaller fragments are combined and tested, is an effective way of designing hit molecule libraries for *Pf*GST.

### 4.1.2 Malaria

Malaria is caused by protozoan organisms, of which *Plasmodium vivax* is the most widespread and *Plasmodium falciparum* is the most lethal. Malaria is carried by the mosquito vector, *Anopheles gambiae,* and has a life cycle that spans the human host and the vector. Malaria is a serious global issue with a third of the world's population exposed to it. An

estimated 515 million clinical cases occur every year (Snow *et al.,* 2005) with up to 2.7 million resulting in death (Murphy & Breman, 2001), making it one of the globe's worst infectious diseases, affecting almost a tenth of the worlds population. The matter is further affected by the fact that 70% of cases occur in Africa where access to Malaria prevention resources and medical treatment is limited. Also of concern is the increasing resistance of malaria strains against current drug treatments. Drug resistance is currently being combated by multi-drug treatments; however, this is by no means a sustainable solution and new drugs still need to be developed (van Vugt *et al.,* 2002).

### 4.1.3 Phases in Malaria Development

During development in the mosquito, gametes fertilize to form a zygote, which then enlarges in the outer wall of the insect's intestine and forms a number of sporozoites (Figure 4.1). The sporozoites move to the salivary glands of the mosquito where they can infect a human host during the mosquito's next blood meal. During development in the human host, the sporozoites enter liver cells and starts replicating in what is know as the schizont phase. Cells known as merozoites are liberated from the liver cells and enter the bloodstream where they infect erythrocytes. During the intraerythrocyte phase replication proceeds as in the liver, however, some of the cells liberated from the erythrocyte, known as gametocytes, do not infect other red blood cells and are only infective to the vector. During a blood meal on an infected human, the mosquito ingests the gametocytes, which then matures into gametes thus completing the lifecycle. Of note to the work being reported here is the intraerythrocyte phase of the malaria lifecycle.

### 4.1.4 Intraerythrocyte Phase of Malaria Lifecycle

At the beginning of the intraerythrocyte phase, a merozoite invades the erythrocyte and becomes enclosed in a parasitophorous vacuole (Figure 4.1), and the infection is said to be in the ring stage. After about 15 hours, the ring cell undergoes an increase in metabolic rate and develops into a trophozoite. The trophozoite continues to grow for approximately 21 more hours at which stage it is 1/3 of the size of the erythrocyte. During the trophozoite stage the plasmodial cell is largely reliant on the cytosolic proteins – primarily haemaglobin – of the

erythrocyte as a source of amino acids (Kamchonwongpaisan *et al.,* 1997). After the trophozoite stage the trophozoite enters the schizont stage and starts replicating into 20-30 daughter merozoites, which are liberated into the blood stream where they can infect more erythrocytes.

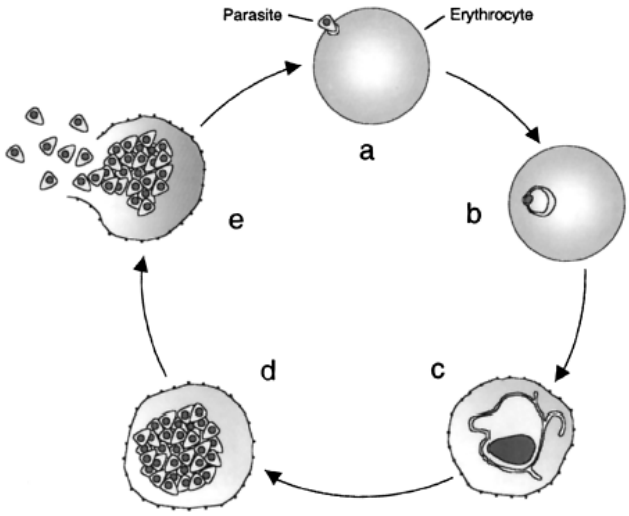



Figure 4.1: Illustration of the different entraerythocyte stages of *Plasmodium falciparum*. (a) Erythrocyte is infected with a single meroziote. (b) The ring stage. (c) The ring stage cell becomes metabolically active, entering the trophozoite stage and growing to about 70% of the erythrocyte's size. (d) The mature trophozoite starts replication, entering the schizont stage, resulting in 20-30 new merozoites. (e) The erythrocyte bursts and the merozoites are liberated into the bloodstream (Kirk, 2001).

### 4.1.4.1 Oxidative Stress

The effect of oxidative stress on Malaria infection has been well documented (Brands *et al.,* 1997; Greve *et al.,* 1999), and it is during the trophozoite stage of the intraerythrocyte phase of the Malaria lifecycle that it has the greatest influence on both the plasmodial cell and the host erythrocyte. The high metabolic rate of the rapidly growing trophozoite leads to the production of large quantities reactive oxygen agents, the large portion resulting from the digestion of haemoglobin (Figure 4.2). The trophozoite endocytosis small portions of erythrocyte cytosol, then fuse it with the acidic food vacuole where the haemoglobin is broken down into amino acids in a process that also produces reactive oxygen agents and toxic free haeme (ferri/ferroprotoporphyrin IX; FP) (Kirk *et al.,* 2001). In addition to the metabolic by-products, the oxidative stress on the cell is further increased by the host immune response.

*Plasmodium falciparum* has developed an efficient system for coping with oxidative stress. The FP in the food vacuole is oxidized from Fe (II) to Fe (III) with the consequent production of superoxide. The trophozoite has a cytosolic Fe-dependent superoxide dismutase; however, it is believed that most of the superoxide is spontaneously transformed to $H_2O_2$ and $O_2$ in the acidic food vacuole (Müller, 2004). Resulting $H_2O_2$ is reduced to $H_2O$ and $O_2$ in the trophozoite by peroxiredoxins, and *Pf*GST in a glutathione (GSH) dependant reaction. GST also has a weak peroxidase activity, but since it constitutes 1-10% of the total cellular protein, it is believed that GST makes a substantial contribution to the total peroxidase activity of the trophozoite (Harwaldt *et al.,* 2002).

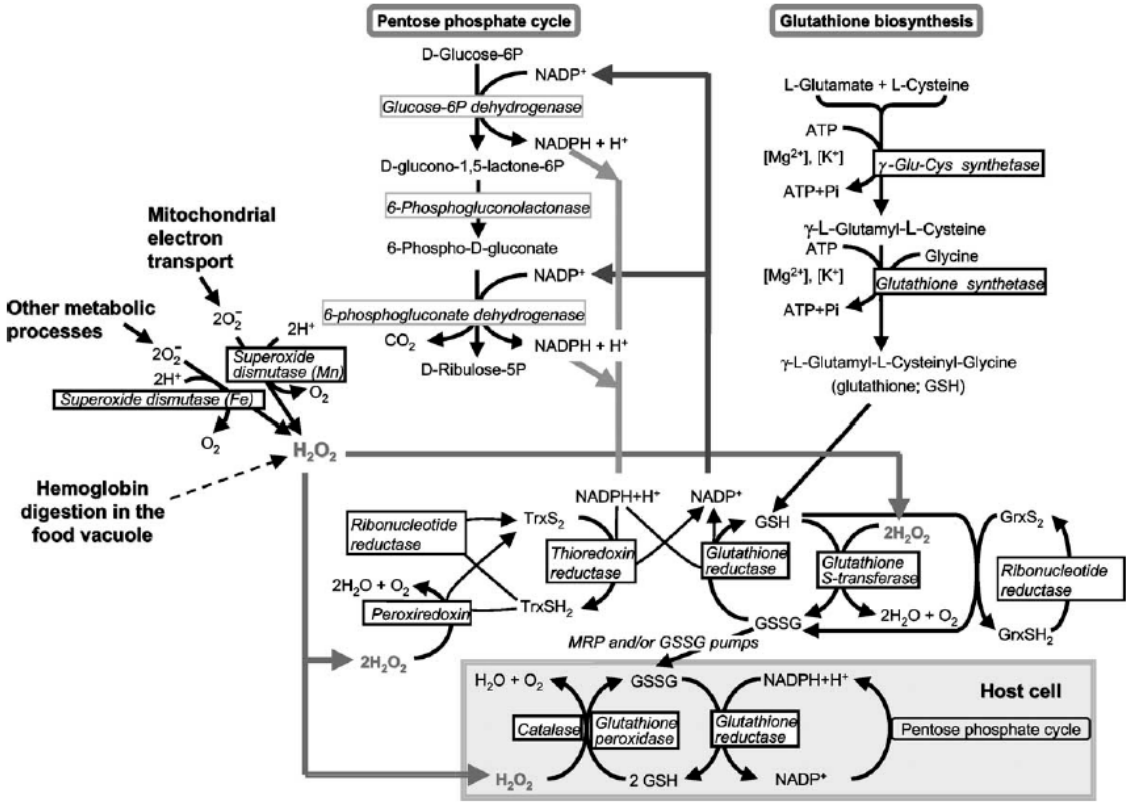



Figure 4.2: Illustration of the metabolic pathways relating to GSH and the oxidative stress response mechanism of *Plasmodium falciparum* (Becker *et al.,* 2004).

### 4.1.4.2 Toxic Effect of Ferri/Ferroprotoporphyrin IX

The toxic FP liberated from haemoglobin is disposed of primarily by sequestering it into the crystaline form called haemozoin while the FP is still in the food vacuole (Egan *et al.,* 2002), and degraded in the presence of GSH (Famin *et al.,* 1999). It is however likely that

some of the FP escapes the food vacuole and enters the trophozoite and erythrocyte cytosol (Becker *et al.,* 2004). FP is extremely toxic, even in low concentrations, and will inhibit enzymes (Campanale *et al.,* 2003) and lyse erythrocytes (Fitch *et al.,* 1982). The parasite protects itself and the host erythrocyte from free FP with the FP-binding proteins Histidine Rich Protein II (HRP2), Histidene Rich Protein III (HRP3) and GST, which acts as a ligandin and buffers the free FP (Choi *et al.,* 1999; Liebau *et al.,* 2002).

### 4.1.5 *Plasmodium falciparum* Glutathione S-Transferase as a Drug Target

#### 4.1.5.1 Glutathione S-Transferase

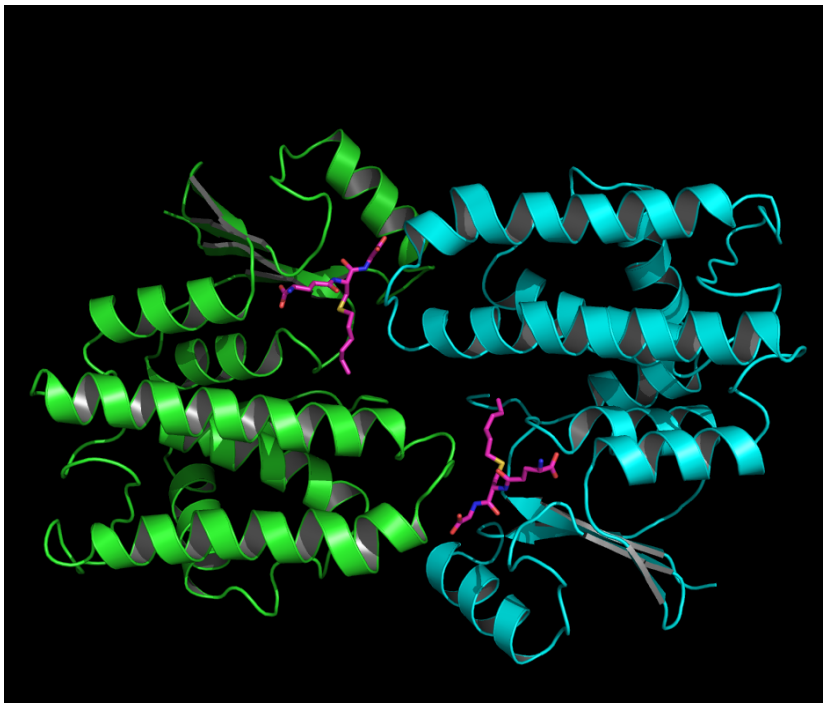


Figure 4.3: The homodimer crystal structure of GST with GSX in the active site. The green chain to the left is the α-subunit and the blue chain to the right the β-subunit. The two purple structures are the GSX molecules.

*Plasmodium falciparum* Glutathione S-transferase (*Pf*GST; Figure 4.3) plays an integral part in detoxification, and maintaining the redox environment, and it does so in both the trophozoite and erythrocyte since the erythrocyte proteins are being digested by the trophozoite thus decreasing the erythrocyte's ability to maintain its cytosolic environment.

*Pf*GST (EC 2.5.1.18) is a functional homodimer of 2.6kDa that catalyses the conjugation reaction of GSH (Figure 4.6) to a large variety of electrophilic substrates, thus labelling it for metabolism and excretion from the cell (Harwaldt *et al.,* 2002; Deponte & Becker, 2005). *Plasmodium falciparum* has a single GST isoenzyme, which is classified as a novel type of GST and its active site differs substantially from its human counterpart, thus making it an excellent drug target (Hiller *et al.,* 2006). The crystal structure of *Pf*GST has been solved for the protein in apoform (Fritz-Wolf *et al.,* 2003; Burmeister *et al.,* 2003) and for the structure with the inhibitor S-hexylglutathione (GSX) in the active site (Perbandt *et al.,* 2004; Hiller *et al.,* 2006). The protein has two active site cavities, the G-site, where GSH binds, and the hydrophobic H-site, where the other conjugation substrates bind. The GSH is oriented in the G-site with its thiol group pointing to the H-site where its sulphur is oxidized to a thiolate, followed by the catalysed nucleophilic attack on electrophilic groups. GST has also been shown to have a small, glutathione dependent peroxidase activity (Harwaldt *et al.,* 2002).

In addition to its catalytic functions, *Pf*GST has also been shown to have the ability to bind hemin in the presence of GSH (Liebau *et al.,* 2002), and to buffer its concentration. Docking studies and assay kinetics done by Liebau *et al.* (2005) show that each subunit of the *Pf*GST homodimer can bind a haeme molecule (Figure 4.4), and it seems that the binding is allosterically activated by GSH binding the G-site, with no haeme binding occurring in the absence of GSH. Interestingly, Liebau also found that when GSH is replaced with S-methylglutathione, haeme cannot bind to GST. This suggests that the deprotonated GSH thiolate plays a crucial role in the intraprotein rearrangement and activation of haeme binding activity of GST.

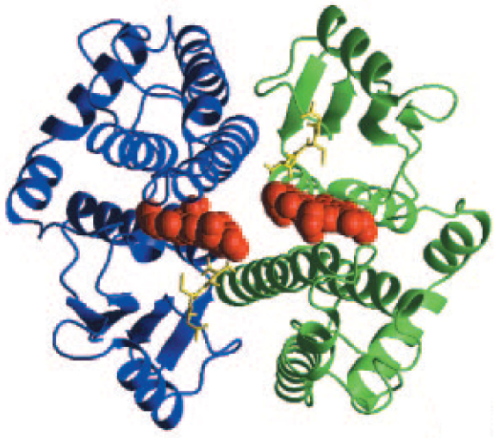


Figure 4.4: The homodimeric structure of *Pf*GST with GSH and haeme docked into both subunits as predicted by Liebau *et al.* (2005).

#### 4.1.5.2 Effects of Targeting Glutathione S-Transferase

The effects of targeting GST as a drug target are three-fold: Firstly, GST inhibition will reduce the conjugation to GSH activity thereby preventing the cell from sequestering harmful molecules such as drugs. Secondly, it will inhibit the peroxidase activity of GST, which will, because of the high number of cellular GST, have a marked influence on the trophozoite's ability to counteract the oxidative stress resulting from the digestion of hemoglobin. Thirdly, in what might be of most interest, it might interfere with the ligandin activity of GST, thereby preventing GST from binding and buffering toxic haeme escaping from the food vacuole after hemoglobin digestion.

#### 4.1.5.3 Targeting Glutathione S-Transferase in Conjunction with Chloroquine

Chloroquine was one of the most effective drugs in the fight against malaria for many years, targeting the intraerythrocytic stage of the infection (Chou *et al.,* 1980), but in recent years its effectiveness has been severely reduced by resistance caused by mutations of *Pf*CRT (Sidhu *et al.,* 2002). Chloroquine acts by binding to liberated haeme, preventing haeme polymerization into hemozoin (Egan *et al.,* 2002), and haeme degradation through its interaction with GSH (Famin *et al.,* 1999). The role of GST in the resistance of cancers to chemotherapeutics, and parasite diseases to drugs, is well documented (Salinas & Wong, 1999). However, GST's involvement in the resistance of *Plasmodium falciparum* to chloroquine is a highly controversial topic with evidence both for GST involvement (Dubios *et al.,* 1995; Srivastava *et al.,* 1999) and against GST involvement (Ferreira *et al.,* 2004; Harwaldt *et al.,* 2002).

It is believed that targeting GST with concurrent treatment with chloroquine will act synergistically, not because it will counter chloroquine resistance, even though it might, but because of the combined effect on the intraerythrocyte malaria parasite. It is hypothesized that targeting the G-site of GST will prevent GSH from binding, thus preventing the allosteric activation of GST haeme ligandin activity, increasing the unbuffered haeme concentrations. The effect of chloroquine in conjunction with the now unbuffered haeme will further increase the antimalarial chloroquine effect of increasing free haeme to toxic levels.

### 4.1.6 AutoDock and AutoDock Tools

AutoDock is a molecular docking tool which is used to find putative binding positions and conformations of ligands in the active sites of target proteins (Morris *et al.,* 1998). AutoDock is the most cited of all docking tools available, being cited 48% of the citation share of the top 5 docking tools, and 27% of all docking tools (Sousa *et al.,* 2006). The high number of citations illustrates that AutoDock is the most widely accepted docking tool in the scientific community. Comparative studies have also shown that AutoDock outperforms other leading docking tools in predicting correct binding conformations (Bursulaya *et al.,* 2003) and as a virtual screening tool (Park *et al.,* 2006).

AutoDock employs flexible ligand docking during the docking process, which allows the ligand to move through different conformations by exploring its torsional degrees of freedom. The latest version of AutoDock, v4, also allows the active site side chains to explore torsional degrees of freedom, thus incorporating both ligand and protein flexibility. However, as v4 is only in beta, it was decided not this version since it has not yet been evaluated and accepted by the scientific community. It was opted to rather use the most recent version before v4, v3.0.5, and henceforth when referring to AutoDock, v3.0.5 is implied.

AutoDock performs the rapid energy evaluations by using precalculated atomic affinity potential grids – calculated by AutoGrid – to reference energies for the atoms in the ligand while the ligand explores its degrees of freedom. The grid is positioned over the protein active site and is made up of probe points where every point has a potential energy for each atom type in the ligand, resulting from the influence of the protein atoms. AutoDock can run three algorithms of which the Lamarckian Genetic Algorithm (LGA) performs the best (Morris *et al.,* 1998). The LGA is a hybrid of a traditional genetic algorithm and a local energy search algorithm. After each step/generation in the genetic algorithm, a local search is done with the resulting ligand conformations and the local energy minimum ligand conformations are found. These “phenotypic” ligand conformations are transformed into the “genotypic” data, which is used for the transformations and mutations in the next generation of genetic algorithm. The phenotypic changes made to the ligands will be inherited by the offspring, hence the name “Lamarckian” genetic algorithm.

The scoring function used by AutoDock while docking, calculates the total free energy change by summing the free energy change contributions of van der Waals, hydrogen bond, electrostatic, torsional and solvation potentials. For each docking result AutoDock gives an inhibition constant, intermolecular energy, internal energy, torsional energy, binding energy and docking energy score. The binding energy score is the sum of the intermolecular and

tortional free-energy scores, while the docking energy score is the sum of the intermolecular and ligand's internal energy scores.

The Scripps institute – developers of AutoDock – has recently developed AutoDock Tools (ADT), a toolkit embedded in the Python Molecular Viewer that allows the scientist to prepare the ligand and protein for docking, to create the AutoGrid and AutoDock input files and to investigate the resulting dockings.

## 4.2 Materials and Method

### 4.2.1 Experimental Design

The G-site of *Pf*GST was targeted, designing inhibitors using LIGLIB and the Markush-like molecular enumeration technique as describe earlier. The fragments used during the enumeration were designed to allow for maximal hydrogen bond interactions and beneficial binding energies, while avoiding fragments that could lead to the inhibitor performing the GSH-like allosteric activation of the haeme ligandin activity. The hit molecules were evaluated and screened based upon their hypothetical inhibition potential, calculated by docking them into the crystal structure of *Pf*GST and comparing their binding energies calculated by the AutoDock scoring function to that of GSH docked back into the crystal structure.

### 4.2.2 Standardized Docking Protocol

To make the scoring function results comparable with one another, it was necessary to follow an exact standardized docking protocol for each of the docking experiments. The protocol used for all dockings and the relevant parameters are discuss here. All parameters not mentioned here should be assumed to be the same as the default values assigned by ADT. Each ligand was prepared for docking by first adding all hydrogens to the mol2 structure, which was necessary to calculate the charges, then loading it into ADT. Before loading the mol2 file into ADT, it was necessary to remove from the file the comment line "# End of record" placed at the end by Corina. ADT has a bug which leads to it trying to interpret the

line as a bond entry (Appendix A). When loading the ligand, ADT automatically assigns Gasteiger charges to the ligand atoms, determines which bonds are rotatable, and removes all nonpolar hydrogens. All rotatable bonds were set to rotatable during docking conformational searching. For the macromolecule the crystal structure 1Q4J of Perbandt *et al.* (2004) was used. First, all nonpolar hydrogen atoms and water molecules were removed from the target PDB file, after which it was loaded into ADT. Next, the grid.gpf AutoGrid parameter file was prepared with ADT. The grid box was centered on the active site G-site of subunit A of the homodimer macromolecule. The number of grid positions in each direction is set to 72, resulting in a total number of 389017 individual atom positions, with the interposition spacing set to 0.247Å. These gridbox settings were large enough for the molecule to rotate fully within the gridbox. Finaly, the dock.dpf AutoDock parameter file was also prepared with ADT. The search algorithm was set to Lamarkian genetic algorithm with the genetic algorithm settings set to: 150 runs, resulting in 150 possible conformations; population size to 50 individuals; and the number of energy evaluations to 1 500 000. For docking parameters: the translation was set to 0.2Å; the quaternation to 5.0°; the torsion to 0.5°; and the RMS cluster tolerance to 0.75Å.

### 4.2.3 Docking Glutathione into *Pf*GST

#### 4.2.3.1 AutoDock Validation

In order to validate the effectiveness of AutoDock in being able to dock glutathione analogs into the G-site of the active site of GST, the crystal structure complex of GTX and GST was redocked, using the standardized docking procedure and parameters, to see if the crystal structure position and conformation of GTX could be recreated.

### 4.2.3.2 Glutathione AutoDock Scoring

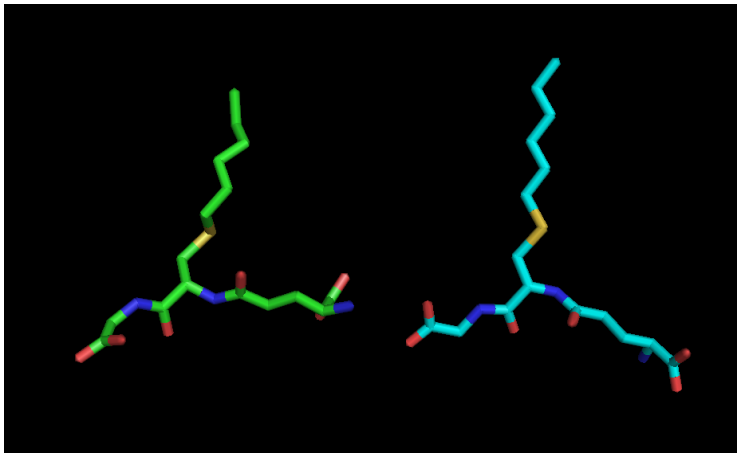


Figure 4.5: The figure illustrates the crystal structure and the Corina generated conformations of GSX. The green structure to the left is the cystal conformation and the blue structure to the left the Corina generated structure. Except for the variable S-hexyl chain, there are only two major changes; the rotation between the α-carbon and nitrogen of the glycine segment; and that of the α-carbon and nitrogen of the cysteine segment.

To put the library screening results into context, it is necessary to have a docking score for the substrate, GSH, with which to compare the screening scores. But for the scores to be comparable, GSH has to be docked under the exact same conditions under which the library was screened. The LIGLIB output molecules have 3D coordinates assigned by Corina, and because Corina coordinates are not always the same as that of crystal structures (Figure 4.5), it was necessary to converted GSH to SMILES format and then back to 3D mol2 format using Corina, giving it coordinates similar to those of the screening library, instead of using the crystal structure of GSX with the hexyl portion removed. The grid.gpf and dock.dpf parameters were made the same as the standardized docking parameters, and the input ligand docking position was made the exact same as those used for the screening library, thereby making the results comparable.

### 4.2.4 Hit Ligand Library Design

#### 4.2.4.1 Library Design with LIGLIB

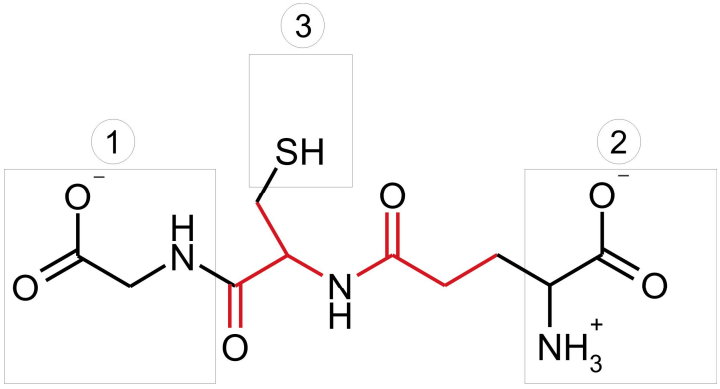



Figure 4.6: Structure of GSH. Each of the three boxes indicates fragments that were removed to create the scaffold, indicated in red. The three boxes also indicate the three positions where LIGLIB made the permutation.

The hit library was built with a single run of LIGLIB, giving a total library of 72 molecules. For LIGLIB to build a library, it requires a set of molecular building blocks: a scaffold; and a range of linkers; and fragments. A reduced (fragments removed from structure) GSH structure was used as scaffold, giving a scaffold that fits well into the active site and that will present the molecular linkers and fragments into the correct protein cavities. GSH is a conjugate of the amino acids glutamate, cysteine and glycine. The structure can be seen as having three outward pointing portions: the glycine end; the α-group of the γ-attached glutamate end; and the thiol group of cysteine. The scaffold was created by removing a fragment at each of the portions positions, leaving a core scaffold with three positions at which permutations could be made (Figure 4.6). Tables 4.1-4.3 presents the molecular building blocks used to build the library, each table representing an individual permutation site. The attachment positions on the fragments and linkers are indicated in red. The fragments building process was based on a rational investigation of the GST active site and its potential interaction positions, specifically, hydrogen bond donors and acceptors (Figure 4.7). Each of the main G-site cavities was investigated in turn, and the fragments built to act as a key within it. The portions removed from positions 1 and 2 to create the scaffold were also used as fragments, but the thiol/thiolate section of position 3 was not used since it is believed that it plays a key role in the allosteric activation of the ligandin activity (Liebau *et al.,* 2005).

At the far end of the permutation site 1 cavity, THR121 was targeted which has a α-nitrogen and a γ-oxygen acting as hydrogen bond donors, while at the end closest to the scaffold LYS117 was targeted, which has a α-oxygen acting as a hydrogen bond acceptor. Permutation site 2 is twice the size of site one, with many more potential interaction residues, all of which were targeted. Potential hydrogen bond donors were: ε-nitrogen of LYS15; α-nitrogen of SER72; and α-nitrogen and δ-nitrogen of GLN71. Potential hydrogen bond acceptors were: δ-oxygen of GLN73; and δ-oxygen of GLN71. Permutation site 3 is positioned on the interface between the G-site and the H-site of the GST active site, is mostly hydrophobic, and only the hydrogen bond acceptor, α-oxygen of PHE116, and the hydrogen bond donor, δ-nitrogen of GLN118, was targeted.

Table 4.1: Fragments used as molecular building blocks at permutation site 1.

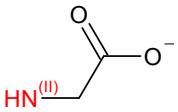
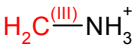
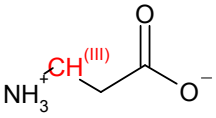
| | Structure | SMILES |
|---|---|---|
| Fragment 1 |  | [NH][CH2][C]([O-])=[O] |
| Fragment 2 |  | [CH2][NH3+] |
| Fragment 3 |  | [CH]([CH2][C]([O-])=[O])[NH3+] |

Table 4.2: Fragments used as molecular building blocks at permutation site 2.

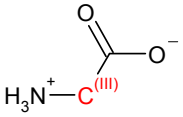
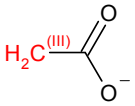
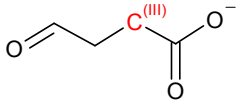
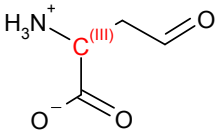
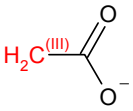
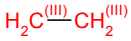
| | Structure | SMILES |
|---|---|---|
| Fragment 1 |  | [CH]([NH3+])[C]([O-])=[O] |
| Fragment 2 |  | [CH2][C]([O-])=[O] |
| Fragment 3 |  | [CH]([C]([O-])=[O])[CH2][CH]=[O] |
| Fragment 4 |  | [C]([NH3+])([C]([O-])=[O])[CH2][CH]=[O] |

Table 4.3: Fragments and linkers used as molecular building blocks at permutation site 3.

| | Structure | SMILES |
|---|---|---|
| Fragment 1 |  | [NH3+] |
| Fragment 2 |  | [CH2][C]([O-])=[O] |
| Fragment 3 |  | [CH]=[O] |
| Linker 1 |  | [CH2] |
| Linker 2 |  | [CH2][CH2] |

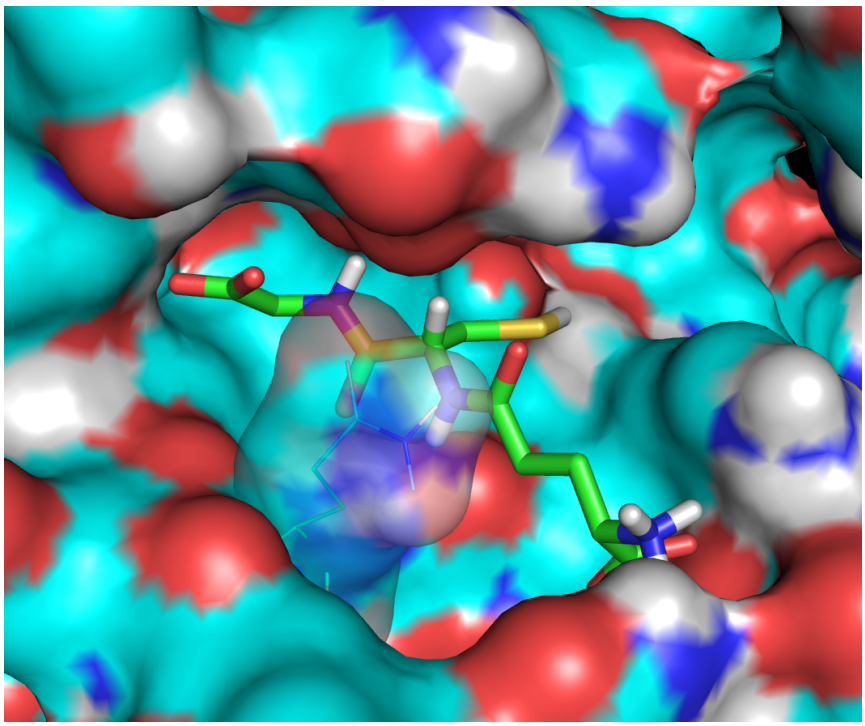


Figure 4.7: Illustration of the G-site portion of the *Pf*GST active site containing GSH.

#### 4.2.4.2 Virtual Screening with AutoDock

The entire library was screened against the *Pf*GST G-site of the alpha subunit of the homodimer. In order to give AutoDock a good position to start the genetic algorithm, each of the ligands needed to have starting coordinates placing each of the three scaffold permutation positions in the correct G-site cavity. The output library coordinates assigned by Corina puts each of the three permutation positions of the scaffold at roughly the same position, and to move the entire library into the correct position, it was simply necessary to move the protein coordinates to fit the library.

## 4.3 Results

### 4.3.1 Standardized Docking Results Evaluation Protocol

All docking results were evaluated using ADT. The docking conformations were visualized in the active site of the target. To simplify the structure being visualized, the protein was viewed with its molecular surface – calculated by MSMS (Sanner & Olson, 1996) – with a 1.5Å radius probe, and its atoms coloured by atom type. Each of the docking results was clustered with a 0.75Å cut-off value, which worked sufficiently for the majority of the dockings. The parameters used in the experiment resulted in 150 docking conformations for each of the ligands docked. Deciding which of the conformations was correct requires the scientist's scrutiny. Initially, the largest clusters in the results were inspected (Figure 4.8). A large cluster around a certain conformation indicated that AutoDock was able to recreate that conformation with separate genetic algorithm runs. Next, the energy scores of the conformations were taken into account. Finally, the docked conformations needed to be investigated in the active. The optimal docking result would be a single, large cluster of conformations with a significant lowering in the system's energy. When reporting a score for the docking, the change in energy value of the highest scoring member of the chosen cluster is reported, as that conformation is the centre of the cluster.

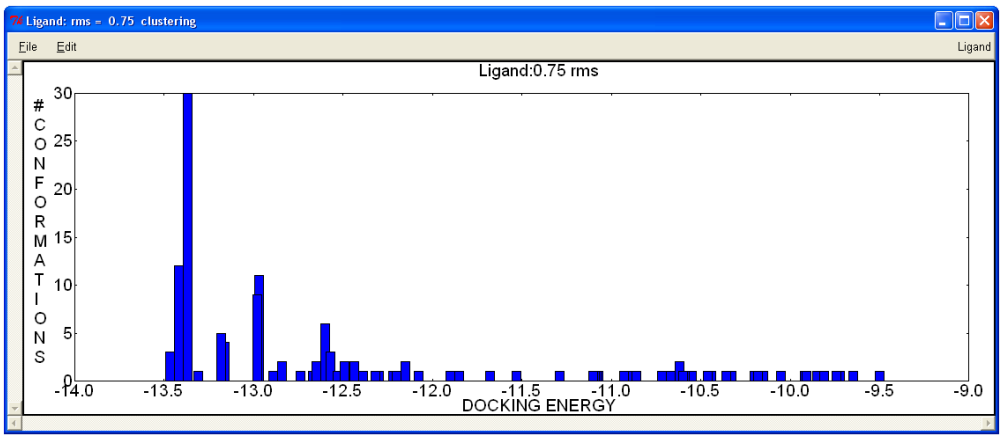



Figure 4.8: The graph generated by ADL representing the clustering of docking results for a Corina-generated glutathione structure. The graph fits to the expected result from a successful docking study: there is a cluster substantially larger than the rest, with a high change in the energy of the system.

### 4.3.2 Docking Glutathione into *Pf*GST

#### 4.3.2.1 AutoDock Validation Results

A common method of determining the performance of a docking tool for docking ligands into a specific macromolecule is to redock a crystal structure ligand and macromolecule and to compare the results to the resulting coordinates to that of the crystal complex (Sousa *et al.,* 2006). Using a RMS cutoff of 1Å, a high scoring cluster of 10 conformations was found of which the average RMS deviation from the crystal conformation coordinates was 0.98Å (Figure 4.9). There are two variable positions in the conformations. The first and most prominent is the S-hexyl chain, and its variability can be explained by the fact that it protrudes the H-site, which is large and highly hydrophobic, allowing the chain to take many conformations during docking with insignificant changes in binding energies. The second is the α-amino group of the γ-glutamate. The α-amino group of the crystal conformation lies extremely close to the γ-amino group of GLN71, and even though it has these coordinates in the crystal structure, it is unlikely that it will spend much time this close to GLN71 in solution. It will rather be positioned as in the docked conformations as they are further from GLN71 and are more energetically favourable (Figure 4.7). The docked conformations are

for the most part capable of creating the same hydrogen bonds as the crystal conformation. The less than 1Å average cluster to crystal coordinate deviation, in spite of the two variable portions, indicates that AutoDock is capable of docking GST analogs into the G-site of GST.

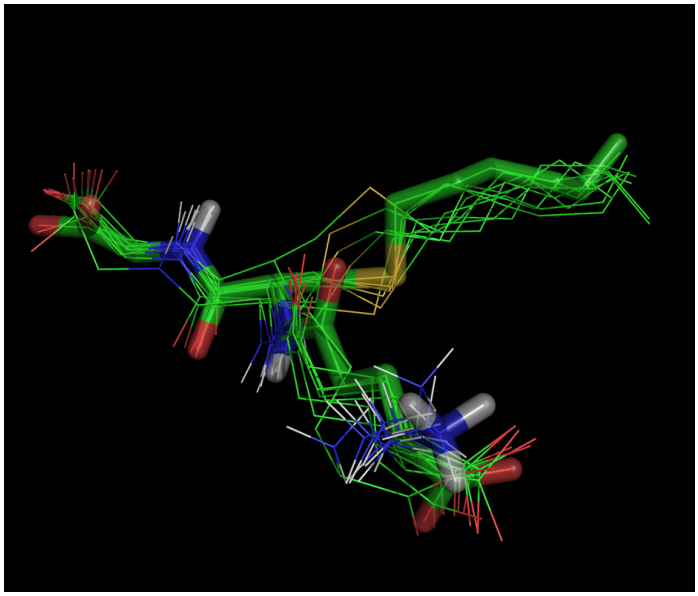


Figure 4.9: A comparison between the crystal structure of GSX and the ten conformations in the result cluster of the AutoDock validation experiment. The partially transparent conformation in stick representation is the crystal structure, while the rest of the conformations in line representation are the docking conformations. There is an obvious similarity in structure, except for the regions mentioned in the text.

#### 4.3.2.2 Glutathione AutoDock Scoring Results

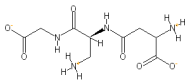
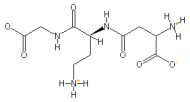
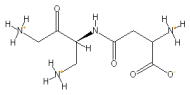
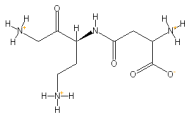
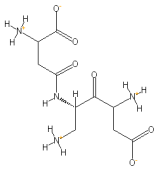
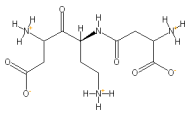
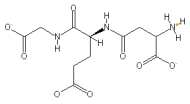
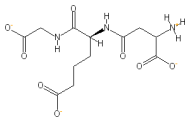
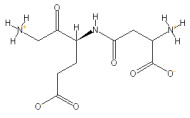
The redocking of GSH under experimental conditions gave a cluster of 30 conformations where the cluster cutoff used was 0.75Å RMS deviation. The best conformation in the cluster has a binding energy score of -13.37 kcal $mol^{-1}$ and a docking energy score of -9.5 kcal $mol^{-1}$.

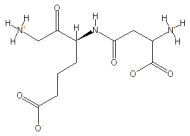
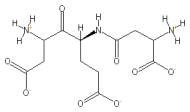
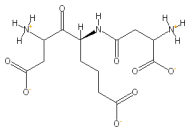
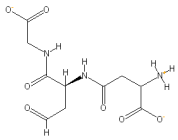
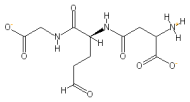
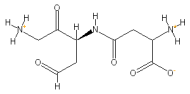
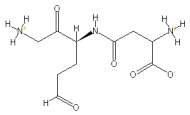
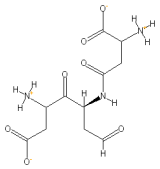
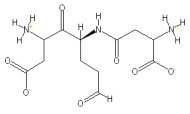
### 4.3.3 Hit Molecule Library Design Results

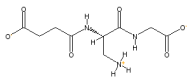
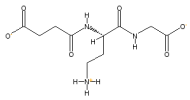
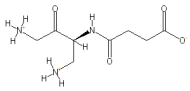
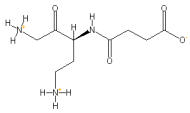
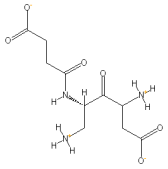
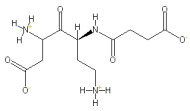
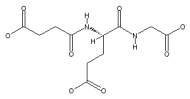
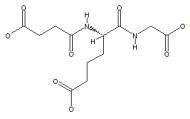
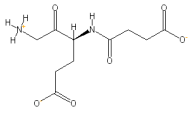
When looking at the 72 molecules in the screening library (Table 4.4), it is obvious that a small, focused library was generated within the localized chemical space around glutathione. Forty-one of the ligands had better docking energy scores (lower scores) than glutathione, which is more than half of the screening library, indicating the good quality of the library. Further more, 16 ligands scored better than -14.5 kcal $mol^{-1}$ and 7 scored better than -15 kcal $mol^{-1}$.

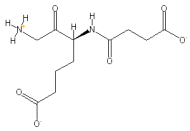
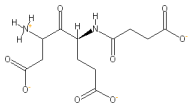
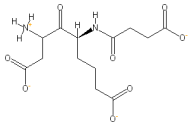
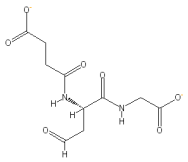
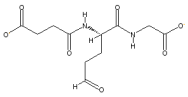
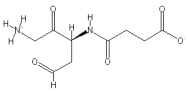
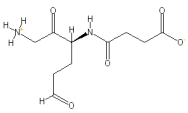
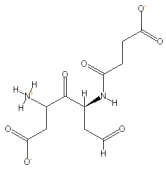
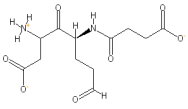
When comparing the prevalence of the molecular building blocks (Tables 4.1-4.3) in the top sixteen scoring ligands, a definite pattern is visible. At permutation position 1, fragment 1 occurred 13 times, while fragment 3 occurred three times and fragment 2 not even once, indicating that fragment 1, which was the original fragment removed from the glutathione scaffold, performed best within the small position 1 active-site cavity. At permutation position 2, fragment 3 was the most prevalent, occurring 8 times, while fragments 1 and 2 occurred three times and fragment 4 twice. Fragment 3 differed from the original fragment removed from the glutathione scaffold, fragment 1, in two ways: it had an acetaldehyde fragment protruding into the portion of the position 2 cavity not filled by glutathione, and did not have the amine group. The negative effect of the amine group is clearly shown by the fact that it is the only difference in the structures of fragment 3 and fragment 4. At position 3, fragment 2 was the most prevalent, occurring ten times, five times with each linker, while fragments 1 and 3 occurred three times each. In most of the cases, the use of linker 2 caused a slightly lower docking energy score than when linker 1 was used, indicating the the extra carbon the in the position 3 linker brings the effector group closer to interaction positions within the active site. Interestingly, when looking at structure 43(the best scoring structure) it is noticeable that at each of its permutation positions, the most prevalent fragment for that position occurs.

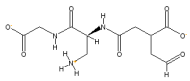
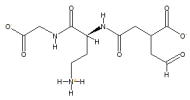
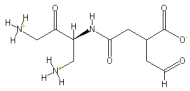
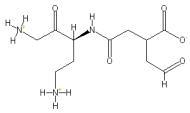
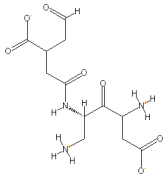
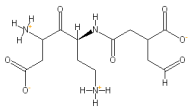
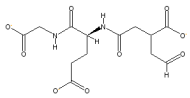
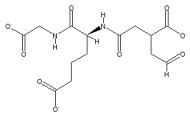
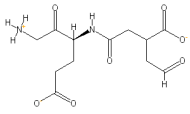
Table 4.4: AutoDock results for the screening library generated by LIGLIB.

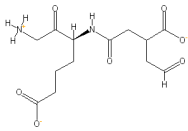
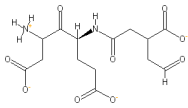
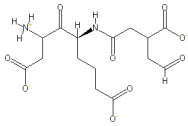
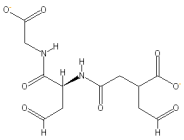
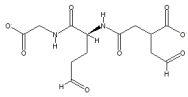
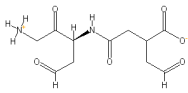
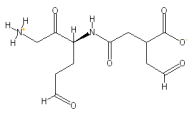
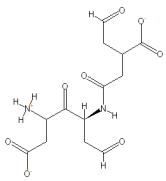
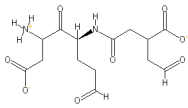
| # | Structure | Cluster Size | Number Clusters | Docking Energy (kcal mol$^{-1}$) | Binding Energy (kcal mol$^{-1}$) |
|---|---|---|---|---|---|
| 1 |  | 31 | 81 | -13.7 | -9.77 |
| 2 |  | 12 | 115 | -11.65 | -7.52 |
| 3 |  | 39 | 77 | -11.91 | -8.83 |
| 4 |  | 10 | 111 | -12.36 | -8.97 |
| 5 |  | 10 | 110 | -13.98 | -11.06 |
| 6 |  | 3 | 129 | -12.26 | -8.7 |
| 7 |  | 11 | 115 | -15.03 | -10.31 |
| 8 |  | 3 | 133 | -15.16 | -10.16 |
| 9 |  | 15 | 111 | -13.16 | -9.32 |

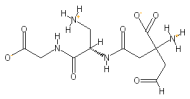
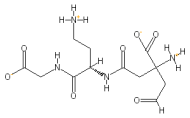
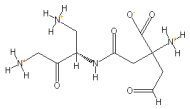
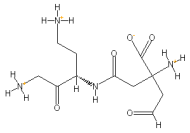
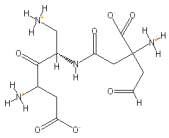
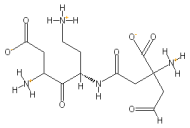
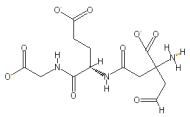
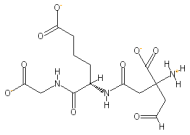
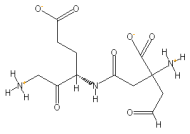
| # | Structure | Cluster Size | Number Clusters | Docking Energy (kcal $mol^{-1}$) | Binding Energy (kcal $mol^{-1}$) |
|---|---|---|---|---|---|
| 10 |  | 9 | 126 | -13.62 | -9.47 |
| 11 |  | 5 | 126 | -13.45 | -9.25 |
| 12 |  | 3 | 143 | -13.73 | -9.63 |
| 13 |  | 16 | 99 | -13.8 | -9.52 |
| 14 |  | 11 | 102 | -14.5 | -9.77 |
| 15 |  | 22 | 93 | -11.91 | -8.5 |
| 16 |  | 14 | 108 | -12.65 | -8.9 |
| 17 |  | 7 | 116 | -12.3 | -8.62 |
| 18 |  | 5 | 129 | -12.39 | -8.22 |

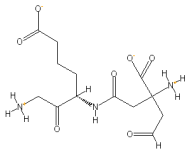
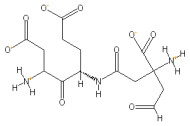
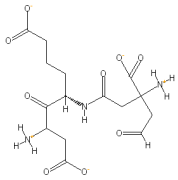
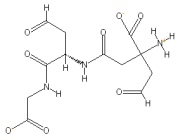
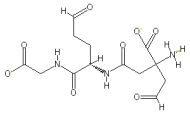
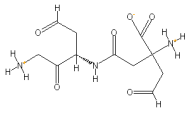
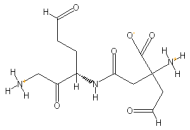
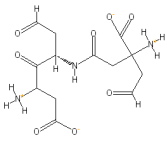
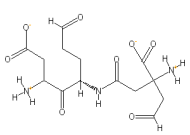
| # | Structure | Cluster Size | Number Clusters | Docking Energy (kcal mol$^{-1}$) | Binding Energy (kcal mol$^{-1}$) |
|---|---|---|---|---|---|
| 19 |  | 56 | 51 | -13.49 | -9.58 |
| 20 |  | 10 | 90 | -13.84 | -9.67 |
| 21 |  | 52 | 48 | -11.69 | -8.65 |
| 22 |  | 9 | 89 | -12.03 | -8.67 |
| 23 |  | 6 | 102 | -13.85 | -10.96 |
| 24 |  | 5 | 121 | -12.68 | -8.82 |
| 25 |  | 27 | 83 | -14.83 | -10.14 |
| 26 |  | 8 | 119 | -15.1 | -10.05 |
| 27 |  | 23 | 88 | -12.13 | -8.79 |

| # | Structure | Cluster Size | Number Clusters | Docking Energy (kcal mol$^{-1}$) | Binding Energy (kcal mol$^{-1}$) |
|---|---|---|---|---|---|
| 28 |  | 13 | 117 | -13.41 | -9.36 |
| 29 |  | 4 | 116 | -15.11 | -11.66 |
| 30 |  | 8 | 127 | -13.88 | -9.41 |
| 31 |  | 35 | 74 | -13.54 | -9.36 |
| 32 |  | 31 | 90 | -14.36 | -9.68 |
| 33 |  | 22 | 76 | -11.78 | -8.24 |
| 34 |  | 37 | 79 | -12.51 | -8.86 |
| 35 |  | 6 | 101 | -13.06 | -9.42 |
| 36 |  | 4 | 120 | -14.05 | -10.04 |

| # | Structure | Cluster Size | Number Clusters | Docking Energy (kcal mol$^{-1}$) | Binding Energy (kcal mol$^{-1}$) |
|---|---|---|---|---|---|
| 37 |  | 19 | 99 | -14.89 | -10.52 |
| 38 |  | 8 | 113 | -15.12 | -10.62 |
| 39 |  | 25 | 91 | -12.93 | -9.54 |
| 40 |  | 5 | 117 | -13.32 | -9.43 |
| 41 |  | 6 | 124 | -14.3 | -10.34 |
| 42 |  | 2 | 140 | -14.96 | -11.71 |
| 43 |  | 7 | 121 | -15.82 | -10.83 |
| 44 |  | 3 | 132 | -14.88 | -10.83 |
| 45 |  | 8 | 112 | -14.41 | -10.64 |

| # | Structure | Cluster Size | Number Clusters | Docking Energy (kcal mol$^{-1}$) | Binding Energy (kcal mol$^{-1}$) |
|---|---|---|---|---|---|
| 46 |  | 2 | 137 | -14.45 | -9.9 |
| 47 |  | 4 | 135 | -13.98 | -9.63 |
| 48 |  | 2 | 138 | -14.98 | -10.3 |
| 49 |  | 10 | 99 | -14.72 | -10.32 |
| 50 |  | 6 | 117 | -15.26 | -10.3 |
| 51 |  | 10 | 83 | -13.08 | -9.49 |
| 52 |  | 6 | 99 | -13.65 | -9.59 |
| 53 |  | 4 | 124 | -13.05 | -9.07 |
| 54 |  | 3 | 125 | -13.63 | -9.26 |

| # | Structure | Cluster Size | Number Clusters | Docking Energy (kcal mol$^{-1}$) | Binding Energy (kcal mol$^{-1}$) |
|---|---|---|---|---|---|
| 55 |  | 17 | 106 | -11.63 | -10.13 |
| 56 |  | 3 | 132 | -13.4 | -10.47 |
| 57 |  | 23 | 102 | -11.54 | -9.63 |
| 58 |  | 8 | 123 | -12 | -9.6 |
| 59 |  | 2 | 134 | -13.49 | -11.25 |
| 60 |  | 3 | 142 | -13.07 | -11.14 |
| 61 |  | 4 | 129 | -14.54 | -11.02 |
| 62 |  | 3 | 133 | -14.65 | -11.13 |
| 63 |  | 13 | 117 | -12.84 | -10.23 |

| # | Structure | Cluster Size | Number Clusters | Docking Energy (kcal mol$^{-1}$) | Binding Energy (kcal mol$^{-1}$) |
|---|---|---|---|---|---|
| 64 |  | 4 | 133 | -13.14 | -10.36 |
| 65 |  | 1 | 141 | -14.4 | -11.58 |
| 66 |  | 2 | 146 | -13.98 | -11.09 |
| 67 |  | 6 | 117 | -13.27 | -10.24 |
| 68 |  | 2 | 131 | -13.88 | -10.4 |
| 69 |  | 37 | 56 | -9.68 | -8.05 |
| 70 |  | 13 | 127 | -12.34 | -9.97 |
| 71 |  | 3 | 142 | -12.7 | -10.2 |
| 72 |  | 2 | 139 | -12.2 | -8.97 |

## 4.4 Discussion

The H-site of *Pf*GST is relatively large, hydrophobic, and is a difficult target for rational drug design. On the other hand, the G-site has three definite cavities with a fair number of hydrogen bond donors and acceptors, making it a good rational design target. It was therefore decided to target only the G-site.

For the docking experiment target, a crystal of GST was used with the GST inhibitor, GTX, in the G-site of the crystal. There are two such crystal structures available (Perbandt *et al.,* 2004; Hiller *et al.,* 2006) of which Perbandt's structure (2.2Å) was obtained by soaking the *Pf*GST crystal with GSH, while Hiller's structure (2.4Å) was obtained by co-crystallizing GSH with *Pf*GST. For this experiment the 2.2Å structure of Perbandt was chosen since it had a better resolution.

When comparing the conformations of both GSX molecules in the crystal structure alpha and beta subunits, virtually the same conformation was found, the only notable change being in conformation in the S-hexyl chain protruding into the H-site. The RMS deviation of all the atoms in the G-site was 0.137Å, indicating that in a 2.2Å crystal structure, it is safe to say that the G-site is virtually the same for both subunits. It is therefore unnecessary to perform docking experiments directed at the G-site, in both subunits of the homodimer crystal structure.

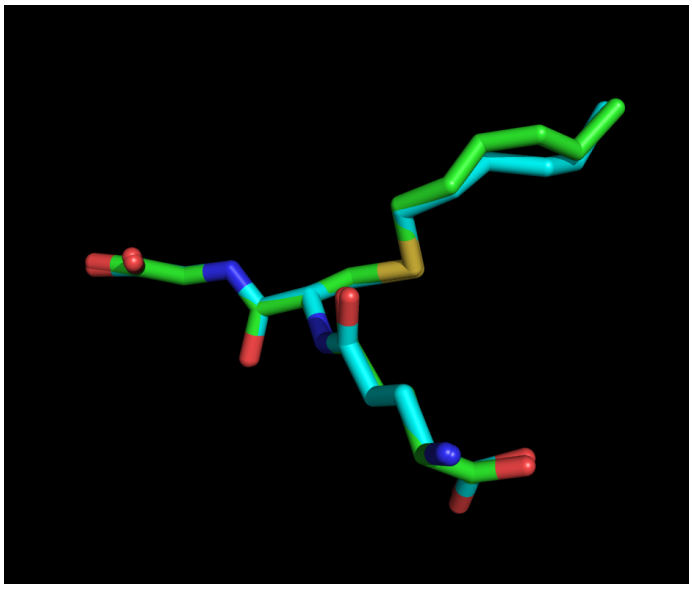


Figure 4.10: The superimposed crystal structure conformations taken by both GSX molecules in the active sites of the α-subunit, blue structure, and the β-subunit, green structure. There is virtually no difference in the G-site portions of the two structures.

In order to keep the results comparable, it was necessary to allow all possible rotatable bonds in the ligand to remain rotatable for each docking experiments. The high number of rotatable bonds increases the variability of the docking conformations, making it more difficult to cluster the results, however, this effect as successfully countered for most of the docking experiments by setting the number of conformations of each docking to 150 and by using the Lamarckian genetic algorithm, which is capable of docking ligands with a higher number of rotatable bonds.

Except for the number of genetic algorithm runs, all the docking parameters are the same as the docking parameters used in an experiment done by Morris *et al.* (1998), the developers of AutoDock, where they did a validation study by docking 30 structurally known protein-ligand complexes with experimentally determined binding constant. Instead of using the 50 genetic algorithm runs as was used in the experiment by Morris, 150 runs were done in order to counter the variability in result conformations and to make sure that the best result cluster would still be observed. The Lamarckian genetic algorithm was used since the same experiment by Morris showed that it performs the best of the AutoDock algorithms.

The prevalence of certain fragments in the library was very interesting, especially the fact that the best scoring structure, number 43, was made up of the most prevalent fragment at each of the permutation position. It seems that fragment prevalence is an excellent, yet easy way to evaluate the effectiveness of certain fragments in generating the library. It is proposed that the prevalence pattern could possibly be investigated in the lead refinement process by using LIGLIB to build libraries recursively. At the end of each library generation, fragments prevalent in the high scoring structures are added to the building blocks of the next library generation, while the non-prevalent fragments are removed. The recursive building of libraries in this fashion would focus in on a specific area of chemical space possibly likely to contain lead molecules.

The overall effectiveness of LIGLIB and the Markush-like molecular enumeration technique is evident from the docking results. More than half of the ligands performed better than glutathione, 16 scored very good, while 7 ligands scored extremely well and are definite candidates for further investigation. For the top 7 ligands to become leads, it will be necessary to perform ADMET screening, and to synthesize them so as to assay them in the lab to measure *in vitro* inhibition and to determine efficacy in *Plasmodium falciparum* cultures.

## 4.5 Conclusion

The fact that *Plasmodium* has a single GST isoform, that it is a new class of GST, that targeting it has a threefold effect, and that it shows huge potential as a target to be used in conjunction with chloroquine, indicates GST is an excellent drug target and further work on it might make a substantial contribution to the fight against malaria.

While investigating the screening library, definite pattern of fragment prevalence was observed in the high scoring structures, and it is hypothesized that the pattern could be exploited using LIGLIB recursively. The proposed method is also an obvious candidate for automation through a genetic algorithm; however, this will remove the intuitive inputs from the user and will suffer from the same issues as the rest of the *de novo* design tools. The possibility of a hybrid technique could, however, be interesting: the user could still be responsible for deciding on a scaffold, its permutation position, and a range of fragments and linkers, allowing for human intuition to start the process, but then the genetic algorithm will take over the process, building generations of libraries, allowing prevalent fragments to survive, and creating new ones by breeding fragments at the same permutation position with each other, navigating the chemical space around and between the fragments that were build through human intuition.

The active-site structure could be used as a guide for the fragment design, however, for targets with known inhibitors it would be possible to follow an alternative fragment generation method where the fragments would be created by “digesting” known inhibitors into sets of fragments for each permutation position on a promising scaffold. The inhibitor design process was, however, not possible for the current investigation since there are very few inhibitors known to targeting the G-site of the *Pf*GST active site.

Using LIGLIB, a high quality, small, focused hit library was successfully produced for *Pf*GST, validating the Markush-like molecular enumeration technique of building rational drug design ligand libraries, the LIGLIB methodology used to implement the technique, and the overall, as well as the design, implementation and overall performance of the LIGLIB toolkit.

**Chapter 5**

# Concluding Discussion

The project consisted of two major sections. Firstly, the drug design tool LIGLIB was developed, and secondly, LIGLIB was used in a rational drug design study targeting *Pf*GST, both of which yielded successful results.

LIGLIB was designed as an expansion to the molecular visualization tool Chimera, and allows scientists to build molecule libraries through the Markush molecular enumeration technique in an intuitive 3D environment. Such an environment did not exist for a free or open source implementation of the technique until LIGLIB was developed.

LIGLIB was developed using accepted development practices and procedures. The design was object-oriented and should not be difficult to understand for other open source developers. The separation of the functional code to the back-end makes LIGLIB very mobile and allows for it to be plugged into virtually any molecular viewer by simply accessing the C++ functionality through the back-end Type interface, and it is indeed hoped that other groups will incorporate it into their open source tools.

A range of difficulties were encountered during the development of LIGLIB, the biggest of which was to create a bijection function that could map atoms in the 3D mol2 file to their same instance in the SMILES string after the conversion of the scaffold structure. This was solved by developing a graph isomorphism algorithm. The algorithm is tree-based and was optimized with various heuristics. In retrospect, coloured graphs would have been used to optimize the algorithm, but since the existing algorithm performs its calculation in a fraction of a second due to the small sizes of the chemical graphs, and because the graph calculation time is but a fraction of the entire time it takes to build the library, it as deemed unnecessary to refactor the algorithm.

It was attempted to re-use and integrate existing software wherever possible. There are already many molecular visualization packages available and it was wholly sensible to use one of them as a vector application, thus it was decided to use Chimera, which was specifically designed to be extensible. The 3D mol2 to SMILES conversion is done by the

open source file conversion package OpenBabel; however, it was necessary to make use of Corina which is a proprietary package to do the 3D coordinate generation for the output library.

The excellent results in the successful *Pf*GST drug design experiment proved that LIGLIB and the Markush molecular enumeration technique was implemented successfully; and also that LIGLIB can potentially be a valuable tool in rational drug design.

*Pf*GST was thoroughly investigated and evidence presented as to why it could be a good drug target in the fight against malaria. In the approach followed, a scaffold was created by removing 3 terminal fragments from glutathione, resulting in a scaffold that perfectly presents permutation positions into the three cavities of the active site. The fragments were designed by investigating each active site cavity, its interaction positions and known interactions between the cavity and the substrate; the fragments built were likely to interact with these interaction sites. By designing only ten fragments and two linkers, it was possible to build a library of 72 molecules, each within the chemical space likely to hold lead molecules.

The results obtained in the *Pf*GST were highly encouraging, and it is evident that at least for *Pf*GST, the Markush enumeration technique holds promise. It will be interesting to repeat the experiment on protein from other families to compare the relative success rate.

Even though the user should create his/her own fragments using scientific knowledge, LIGLIB's appeal might be increased if it was distributed with a standard library of fragments from which users can select permutation fragments. We also believe that it would be worthwhile to continue developing further functionalities, perhaps even a ligand library generation system, which makes use of various library generation techniques.

The project as a whole was regarded as successful as we have succeeded in all three of the goals set (Section 1.5).

# Summary

The current costs of drug discovery are extremely high and need to be addressed if diseases such as AIDS and malaria are to be combated. The major reasons for the high costs are the use of expensive *in vitro* methods and the high failure rate of drugs at clinical testing phases. *In silico* techniques hold tremendous potential in addressing the high costs. *In silico* drug design can be done at a fraction of the cost of *in vitro* techniques, and can be used in synergy with *in vitro* techniques by doing much of the screening before any experimental studies, thereby reducing the chemical space to be searched experimentally. *In silico* techniques can also enhance the quality of drug candidates sent to clinical phases, increasing the probability of success.

In this study techniques were investigated to build analogous ligand libraries with scaffolds and molecular building blocks through a user guided process, including the development of the LIGLIB program,, which is an Open Source package for lead development. The Markush molecular enumeration technique was implemented in C++ with a Python front-end extending it to the Python molecular visualization tool, Chimera. The software makes use of chemical graphs to make permutations according to user inputs, generating an output library in SMILES and Mol2 format, the later of which is generated by Corina.

As part of the validation of the software it was used in a lead discovery experiment which targeted *Plasmodium falciparum* Glutathione S-transferase. The developed software was able to generate a series of suitable molecules, thereby validating the Markush molecular enumeration technique as well as its implementation in LIGLIB.

# References

Adams C.P. & Brantner V.V. (2006) Estimating the cost of new drug development: Is it really $802 million? *Health Affairs,* **25:** 420 – 428.

Becker K., Tilley L., Vennerstrom, J.L., Roberts D., Rogerson S. & Ginsburg, H. (2004) Oxidative stress in malaria parasite-infected erythrocytes: host parasite interactions. *International Journal for Parasitology,* **34:** 163 – 189.

Blundell T.L., Jhoti H. & Abell C. (2002) High-throughput crystallography for lead discovery in drug design. *Nature Reviews: Drug Discovery,* **1:** 45 – 54.

Blundell T.L. & Patel S. (2004) High-throughput X-ray crystallography for drug discovery. *Current Opinions in Pharmacology,* **4:** 490 – 496.

Bohacek R., McMartin C. & Guida W. (+96) The art and practice of structure-based drug design: a molecular modeling perspective. *Medicinal Research Reviews,* **16:** 3 – 50.

Brandts C.H., Ndjavé M., Graninger W. & Kremsner P.G. (1997) Effect of paracetamol on parasite clearance time in *Plasmodium falciparum* malaria. *The Lancet,* **350:** 704 – 709.

Burmeister C., Perbandt M., Betzel C., Waltera R.D. & Liebaua E. (2003) Crystallization and preliminary X-ray diffraction studies of the glutathione S-transferase from *Plasmodium falciparum*. *Acta Crystallographica Section D,* **59:** 1469 – 1471.

Bursulaya B.D., Totrov M., Abagyan R. & Brooks C.L. 3rd (2003) Comparative study of several algorithms for flexible ligand docking. *Journal of Computer-Aided Molecular Design,* **17:** 755 – 763.

Bywater R.P., Poulsen T.A., Rgen P. & Hjorth P.G. (2004) *De novo* generation of molecular structures using optimization to select graphs on a given lattice. *Journal of Chemical Information and Computer Science,* **44:** 856 – 861.

Campanale N., Nickel C., Daubenberger C.A., Wehlan D.A., Gorman J.J., Klonis N., Becker K. & Tilley L. (2003) Identification and characterization of heme-interacting proteins in the malaria parasite, *Plasmodium falciparum*. *The Journal of Biological Chemistry,* **278:** 27354 – 27361.

Carr R. & Jhoti H. (2002) Structure-based screening of low-affinity compounds. *Drug Discovery Today,* **7:** 522 – 527.

Chen J., Swamidass S.J., Dou Y., Bruand J. & Baldi P. (2005) ChemDB: a public database of small molecules and related chemoinformatics resources. *Bioinformatics,* **21:** 4133 – 4139.

Choi, C.Y.H., Cerda J.F., Chu H., Babcock G.T. & Marletta M.A. (1999) Spectroscopic characterization of the heme-binding sites in *Plasmodium falciparum* histidine-rich protein 2. *Biochemistry,* **38:** 16916 – 16924.

Chou A.C., Chevli R. & Fitch C.D. (1980) Ferriprotoporphyrin IX fulfills the criteria for identification as the chloroquine receptor of malaria parasites. *Biochemistry,* **19:** 1543 – 1549.

Colmenarejo G. (2005) *In Silico* ADME prediction: data sets and models. *Current Computer-Aided Drug Design,* **1:** 365 – 376.

Combrinck J.M., Egan J., Hearne G.R., Marques H.M., Ntenteni S., Sewell B.T., Smith P.J., Taylor D., Schalkwyk D.A.V. & Walden J.C. (2002) Fate of haem iron in the malaria parasite *Plasmodium falciparum*. *Biochemistry,* **365:** 343 – 347.

Cottom T.L. (2003) Using swig to bind c++ to Python. *Computing in Science and Engineering,* **5:** 88 – 97.

Dekker F.J., Koch M.A. & Waldmann H. (2005) Protein structure similarity clustering (PSSC) and natural product structure as inspiration sources for drug development and chemical genomics. *Current Opinions in Chemical Biology,* **9:** 232 – 239.

DeLano W.L. (2005) The case for open-source software in drug discovery. *Drug Discovery Today,* **10:** 213 – 217.

Deponte M. & K.Becker (2005) Glutathione S-transferase from malaria parasites: structural and functional aspects. *Methods in Enzymology,* **401:** 241 – 253.

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.

Dolle R.E., Bourdonnec B.L., Morales G.A., Moriarty K.J. & Salvino J.M. (2006) Comprehensive survey of combinatorial library synthesis. *Journal of Combinatorial Chemistry,* **8**, 597 – 635.

Dubois V.L., Platel D.F.N., Pauly G. & Duret J.T. (1995) *Plasmodium berghei*: Implecation of intracellular glutathione and its related enzyme in chloroquine resistance in Vivo. *Experimental Parasitology,* **81:** 117 – 124.

Erlanson D.A., Wells J.A. & Braisted A.C. (2004) Tethering: fragment-based drug discovery. *Annual Review of Biophysics Biomolecular Structure,* **33:** 199 – 223.

Famin O., Krugliak M. & Ginsburg H. (1999) Kinetics of inhibition of glutathione-mediated degradation of ferriprotoporphyrin IX by antimalarial drugs. *Biochemical Pharmacology,* **58:** 59 – 68.

Ferreira I.D., Nogueira F., Borges S.T., do Rosario V.E. & Cravo P. (2004) Is the expression of genes encoding enzymes of glutathione (GSH) metabolism involved in chloroquine resistance in *Plasmodium chabaudi* parasites? *Molecular and Biochemical Parasitology,* **136:** 43 – 50.

Feuston B.P., Chakravorty S.J., Conway J.F., Culberson, J.C., Forbes J., Kraker B., Lennon P.A., Lindsley C., McGaughey G.B., Mosley R., Sheridan R.P., Valenciano M. & Kearsley S.K. (2005) Web enabling technology for the design, enumeration, optimization and tracking of compound libraries. *Current Topics in Medicinal Chemistry,* **5:** 773 – 783.

Fitch C.D., Chevli R., Banyal H.S., Phillips G., Pfaller M.A. & Krogstad D.J. (1982) Lysis of *Plasmodium falciparum* by ferriprotoporphyrin IX and a chloroquine-ferriprotoporphyrin IX complex. *Antimicrobial Agents Chemotherapy,* **21:** 819 – 822.

Gagliardi F., Jones B., Grey F., Bégin M. & Heikkurinen M. (2005) Building an infrastructure for scientific Grid computing: status and goals of the EGEE project. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* **363:** 1729 – 1742.

Geldenhuys W.J., Gaasch K.E., Watson M., Allen D.D. & der Schyf C.J.V. (2006) Optimizing the use of open-source software applications in drug discovery. *Drug Discovery Today,* **11:** 127 – 132.

Ghosh S., Nie A., An J. & Huang Z. (2006) Structure-based virtual screening of chemical libraries for drug discovery. *Current Opinions in Chemical Biology,* **10:** 194 – 202.

Gillet V.J., Newell W., Mata P., Myatt G., Sike S., Zsoldos Z. & Johnson A.P. (1994) SPROUT is a computer program for constrained structure generation. *Journal of Chemical Informatics and Computer Science,* **34**, 207 – 217.

Goodford P.J. (1985) A computational procedure for determining energetically favorable binding sites. *Journal of Medicinal Chemistry,* **28:** 849 – 857.

Greve B., Lehman L.G., Lell B., Luckner D., Ott R.S. & Kremsner P.G. (1999) High oxygen radical production is associated with fast parasite clearance in children with *Plasmodium falciparum* malaria. *The Journal of Infectious Diseases,* **179:** 1584 – 1586.

Griffith R., Luu T.T., Garner J. & Keller P.A. (2005) Combining structure-based drug design and pharmacophores. *Journal of Molecular Graphics and Modeling,* **23:** 439 – 446.

Guha R., Howard M.T., Hutchison G.R., Rust P.M., Rzepa H., Steinbeck C., Wegner J.K. & Willighagen E.L. (2006) The blue obelisk-interoperability in chemical informatics. *Journal of Chemical Information and Modeling,* **46:** 991 – 998.

Haigh J.A., Pickup B.T., Grant J.A. & Nicholls A. (2005) Small molecule shape-fingerprints. *Journal of Chemical Information and Modeling,* **45:** 673 – 684.

Hardy L.W. & Malikayil A. (2003) The impact of structure-guided drug design on clinical agents. *Current Drug Discovery,* **3:** 15 – 20.

Harper G., Bravi G.S., Pickett S.D., Hussain J. & Green D.V.S. (2004) The reduced graph descriptor in virtual screening and data-driven clustering of high-throughput screening data. *Journal of Chemical Information and Modeling,* **44:** 2145 – 2156.

Harwaldt P., Rahlfs S. & Becker K. (2002) Glutathione S-transferase of the malaria parasite *Plasmodium falciparum*: Characterization of a potential drug target. *Biological Chemistry,* **383:** 821 – 830.

Hiller N., Wolf K.F., Deponte M., Wende W., Zimmermann H. & Becker K. (2006) *Plasmodium falciparum* glutathione S-transferase – Structural and mechanistic studies on ligand binding and enzyme inhibition. *Protein Science,* **15:** 281 – 289.

Humphrey W., Dalke A. & Schulten K. (1996) VMD: Visual Molecular Dynamics. *Journal of Molecular Graphics,* **14:** 33 – 38.

Hutter M.C. (2007) Separating drugs from nondrugs: A statistical approach using atom pair distributions. *Journal of Chemical Information and Modeling,* **47:** 186 – 194.

Irwin J.J. & Shoichet B. (2005) ZINC – a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling,* **45:** 177 – 182.

Jacq F., Salzemann J., Legré Y., Medernach E., Montagnat J., Maass A., Reichstadt M., Schwichtenberg H., Sridhar M., Kasam V., Zimmermann M., Hofmann M. & Breton V. (2007) Grid enabled virtual screening against malaria. T*o appear in Journal of Grid Computing,*

Jónsdóttir S.&., Jorgensen F.S. & Brunak S. (2005) Prediction methods and databases within chemoinformatics: emphasis on drugs and drug candidates *Bioinformatics,* **21:** 2145 – 2160.

Kamchonwongpaisan S., Samoff E. & Meshnick S. (1997) Identification of hemoglobin degradation products in *Plasmodium falciparum*. *Molecular and Biochemical Parasitology,* **86:** 179 – 186.

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

Kirk K. (2001) Membrane transport in the malaria-infected erythrocyte. *Physiological Reviews,* **81:** 495 – 537.

Krier M., Junior J.X.A., Schmitt M., Duranton J., Basaran H.J., Lugnier C., Bourguignon J.J. & Rognan D. (2005) Design of small-sized libraries by combinatorial assembly of linkers and functional groups to a given scaffold: application to the structure-based optimization of a phosphodiesterase 4 inhibitor. *Journal of Medicinal Chemistry,* **48:** 3816 – 3822.

Kubinyi H. (2003) Drug research: myths, hype and reality. *Nature Reviews: Drug Discovery,* **2:** 665 – 668.

Lapinsh M., Prusis, P., Uhlen S. & Wikberg J.E.S. (2005) Improved approach for proteochemometrics modeling: application to organic compound – amine G protein-coupled receptor interactions. *Bioinformatics,* **21:** 4289 – 4296.

Leach A.R. & Hann M.M. (2000) The in silico world of virtual libraries. *Drug Discovery Today,* **5:** 326 – 336.

Leland B.A., Christie B.D., Nourse J.G., Grier D.L., Carhart R.E., Maffett T., Welford S.M. & Smith D.H. (1997) Managing the combinatorial explosion. *Journal of Chemical Informatics and Computer Science,* **37:** 62 – 70.

Liao C., Liu B., Shi L., Zhou J. & Lu X. (2005) Construction of a virtual combinatorial library using SMILES strings to discover potential structure-diverse PPAR modulators *European Journal of Medicinal Chemistry,* **40:** 632 – 640.

Liebau E., Bergmann B., Campbell A.M., Spittle P.T., Brophy P.M., Luersen K. & Walter R.D. (2002) The glutathione S-transferase from *Plasmodium falciparum*. *Molecular & Biochemical Parasitology,* **124:** 85 – 90.

Liebau E., Maria F.D., Burmeister C., Perbandt M., Turella P., Antonini G., Federici G., Giansanti F., Stella L., Bello M.L., Caccuri A.M. & Ricci G. (2005) Cooperativity and pseudo-cooperativity in the glutathione S-transferase from *Plasmodium falciparum. Journal of Biological Chemistry,* **280:** 26121 – 26128.

Lipinski C.A., Lombardo F., Dominy, B.W. & Feeney P.J. (2001) Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced Drug Delivery Reviews,* **1:** 3 – 26.

Lombardino J.G. & Lowe J.A. (2004) The role of the medicinal chemist in drug discovery? Then and now. *Nature Reviews: Drug Discovery,* **3:** 853 – 862.

Marris E. (2005) Chemistry society goes head to head with NIH in fight over public database. *Nature,* **435:** 718 – 719.

Martin Y.C. (1997) Challenges and prospects for computational aids to molecular diversity. *Perspecives in. Drug Discovery and Design,* **7:** 159 – 172.

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.

Mizutani M.Y. & Itai A. (2004) Efficient method for high-throughput virtual screening based on flexible docking: discovery of novel acetylcholinesterase inhibitors. *Journal of Medicinal Chemistry,* **47:** 4818 – 4828.

Moore J., Manan N.A. Fejzo J., Jacobs M., Lepre C., Peng J. & Xie X. (2004) Leveraging structural approaches: applications of NMR-based screening and X-ray crystallography for inhibitor design. *Journal of Synchrotron Radiation,* **11:** 97 – 100.

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.

Murphy S.C. & Breman J.G. (2001) Gaps in the childhood malaria burden in Africa: cerebral malaria, neurological sequelae, anemia, respiratory distress, hypoglycemia, and complications of pregnancy. *American Journal of Tropical Medicine and Hygiene,* **64:** 57 – 67.

Nören-Müller N., Reis-Corrêa I. Jr, Prinz H., Rosenbaum C., Saxena K., Schwalbe H.J., Vestweber D., Cagna G., Schunk S., Schwarz O., Schiewe H. & Waldmann H. (2006) Discovery of protein phosphatase inhibitor classes by biology-oriented synthesis. *Proceedings of the National Acadamy of Sciences,* **103:** 10606 – 10611.

Müller S. (2004) Redox and antioxidant systems of the malaria parasite *Plasmodium falciparum*. *Molecular Microbiology,* **53:** 1365 – 2958.

Müller S., Gilberger T.W., Krnajski Z., Lüersen K., Meierjohann S. & Walter R.D. (2001) Thioredoxin and glutathione system of malaria parasite *Plasmodium falciparum. Protoplasma,* **217:** 43 – 49.

Nienaber V.L., Richardson P.L., Klighofer V., Bouska J.J., Giranda V.L. & Greer J. (2000) Discovering novel ligands for macromolecules using X-ray crystallographic screening. *Nature Biotechnology,* **18:** 1105 – 1108.

Park H., Lee J. & Lee S. (2006) Critical assessment of the automated AutoDock as a new docking tool for virtual screening. *Proteins,* **65:** 143 – 747.

Patrick G. (2005) An introduction to medicinal chemistry, *Oxford University Press, Great Clarendon Street, Oxford OX2 6DP 3rd edition, Chapter 14: Combinatorial Synthesis,* 299 – 325.

Perbandt M., Burmeister C., Walter R.D., Betzel C. & Liebau E. (2004) Native and inhibited structure of a Mu class-related glutathione S-transferase from *Plasmodium falciparum*. *Journal of Biological Chemistry,* **279:** 1336 – 1342.

Pettersen E.F., Goddard T.D., Huang C.C., Couch G.S., Greenblatt D.M., Meng E.C. & Ferrin T.E. (2004) UCSF Chimera – A visualization system for exploratory research and analysis. *Journal of Computational Chemistry,* **25:** 1605 – 1612.

Raymond J.W., Gardiner E.J. & Willett P. (2002) RASCAL: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal,* **45:** 631 – 644.

Raymond J.W. & Willett P. (2002) Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design,* **16**: 521 – 533.

des Rivieres J. & Wiegand J. (2004) Eclipse: A platform for integrating development tools. *IBM Systems Journal,* **43.**

Rupasinghe C.N. & Spaller M.R. (2006) The interplay between structure-based design and combinatorial chemistry. *Current Opinion in Chemical Biology,* **10:** 188 – 193.

Salinas A.E. & Wong M.G. (1999) Glutathione S-transferases – A review. *Current Medicinal Chemistry,* **6:** 279 – 309.

Samiulla D.S., Vaidyanathan V.V., Arun P.C., Balan G., Blaze M., Bondre S., Chandrasekhar G., Gadakh A., Kumar R., Kharvi G., Kim H.O., Kumar S., Malikayil J.A., Moger M., Mone M.K., Nagarjuna P., Ogbu C., Pendhalkar D., Rao A.V., Rao G.V., Sarma V.K., Shaik S., Sharma G.V., Singh S., Sreedhar C., Sonawane R., Timmanna U. & Hardy L.W. (2005) Rational selection of structurally diverse natural product scaffolds with favorable ADME properties for drug discovery. *Molecular Diversity,* **9:** 131 – 139.

Sanner M.F. & Olson A.J. (1996) Reduced surface: An efficient way to compute molecular surfaces. *Biopolymers,* **38:** 305 – 320.

Schneider G. & Fechner U. (2005) Computerbased *de novo* design of drug-like molecules. *Nature Reviews: Drug Discovery,* **4:** 649 – 663.

Schönberger H., Schwab C.H., Hirsch A. & Gasteiger J. (2000) Molecular of fullerene dendrimers. *Journal of Molecular Modeling,* **6:** 379 – 395.

Schüller A., Schneider G. & Byvatov E. (2003) SMILIB: Rapid assembly of combinatorial libraries in SMILES notation. *QSAR & Combinatorial Science,* **22:** 719 – 721.

Sidhu A.B.S., Pinard D.V. & Fidock D.A. (2002) Chloroquine resistance in P*lasmodium falciparum* malaria parasites conferred by *Pf*CRT mutations. *Science,* **298:** 210 – 213.

Singh M.K., Srivastava S., Raghava G.P.S. & Varshney G.C. (2006) HaptenDB: a comprehensive database of haptens, carrier proteins and anti-hapten antibodies. *Bioinformatics,* **22:** 253 – 255.

Snow R.W., Guerra C.A., Noor A.M., Myint H.Y. & Hay S.I. (2005) The global distribution of clinical episodes of *Plasmodium falciparum* malaria. *Nature,* **434:** 214 – 217.

Song H., Wang R., Wang S. & Lin J. (2005) A low-molecular-weight compound discovered through virtual database screening inhibits Stat3 function in breast cancer cells. *Science,* **102:** 4700 – 4705.

Sousa S.F., Fernandes P.A. & Ramos M.J. (2006) Protein-ligand docking: Current status and future challenges. *Proteins,* **65:** 15 – 26.

Srivastava P., Puri S.K., Kamboj K.K. & Pandey V.C. (1999) Glutathione-S-transferase activity in malarial parasites. *Tropical Medicine & International Health,* **4**, 251 – 254.

Strausberg R.L. & Schreiber S.L. (2003) From knowing to controlling: A path from genomics to drugs using small molecule probes. *Science,* **300:** 294 – 295.

Swayze E.E., Jefferson E.A., Lowery K.A.S., Blyn L.B., Risen L.M., Arakawa S., Osgood S.A., Hofstadler S.A. & Griffey R.H. (2002) SAR by MS: A ligand based technique for drug lead discovery against structured RNA targets. *Journal of Medicinal Chemistry,* **45:** 3816 – 3819.

van Vugt M., Leonardi E., Phaipun L., Slight T., Thway K.L., McGready R., Brockman A., Villegas L., Looareesuwan S., White N.J. & Nosten F. (2002) Treatment of uncomplicated multidrug-resistant falciparum malaria with artesunate-atovaquone-proguanil. *Clinical Infectious Diseases,* **35:** 1498 – 1504.

Wang R., Gao Y. & Lai L. (2000) LigBuilder: A multi-purpose program for structure-based drug design. *Journal of Molecular Modeling,* **6:** 498 – 516.

van de Waterbeemd H. & Gifford E. (2003) ADMET in silico modelling: towards prediction paradise? *Nature Reviews: Drug Discovery,* **2:** 192 – 204.

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

Willett P. (2003) Similarity-based approaches to virtual screening. *Biochemical Society Transcripts,* **31:** 603 – 606.

Williams M. (2003) Target validation. *Current Opinions in Pharmacology,* **3:** 571 – 577.

Wishart D.S., Knox C., Guo A.C., Shrivastava S., Hassanali M., Stothard P., Chang Z. & Woolsey J. (2006) DrugBank: a comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Research,* **34:** 668 – 672.

Wolf K.F., Becker A., Rahlfs S., Harwaldt P., Schirmer R.H., Kabsch W. & Becker K. (2003) X-ray structure of glutathione S-transferase from the malarial parasite *Plasmodium falciparum*. *Biochemistry,* **100:** 66045 – 67582.

# Appendix A

```
#       Name:                   S-hexylglutathione
#       Creating user name:     User
#       Creation time:          Thu Jan 04 14:23:14 2007

@<TRIPOS>MOLECULE
S-hexylglutathione
   54    53     1     0     0
SMALL
USER_CHARGES

File created by VEGA ZZ 2.0.7
@<TRIPOS>ATOM
      1 CA1          0.0580    7.8880   -5.5950 C.3       1 UNK1         0.2057
      2 CA2          1.9750   11.8810   -1.1520 C.3       1 UNK1         0.0967
      3 CA3          0.9380   15.1120    0.5200 C.3       1 UNK1         0.0892
      4 CB1          0.3640    8.2490   -4.1420 C.3       1 UNK1        -0.0067
      5 CB2          3.5070   11.8560   -1.1160 C.3       1 UNK1         0.0177
      6 CG1          0.8090    9.6990   -4.0290 C.3       1 UNK1         0.0233
      7 SG2          4.2480   10.2190   -1.4500 S.3       1 UNK1        -0.1587
      8 CD1          0.9940   10.1190   -2.5920 C.2       1 UNK1         0.2014
      9 OE1          0.7650    9.2420   -1.5560 O.2       1 UNK1        -0.2795
     10 C1           1.2680    7.9280   -6.4830 C.2       1 UNK1         0.2075
     11 C1S          3.6720    9.3650    0.0480 C.3       1 UNK1        -0.0063
     12 N1          -0.5650    6.5750   -5.6570 N.4       1 UNK1        -0.0623
     13 C2           1.4990   13.2980   -1.0090 C.2       1 UNK1         0.2193
     14 C2S          4.1080    7.9030    0.0870 C.3       1 UNK1        -0.0440
     15 N2           1.4330   11.3710   -2.3920 N.am      1 UNK1        -0.2462
     16 O2           1.1900   14.0380   -2.1360 O.2       1 UNK1        -0.2778
     17 C3          -0.4250   15.1730    1.1410 C.2       1 UNK1         0.1910
     18 C3S          5.5570    7.8010    0.4960 C.3       1 UNK1        -0.0523
     19 N3           1.3800   13.7640    0.2190 N.am      1 UNK1        -0.2466
     20 C4S          6.2800    6.6520   -0.1890 C.3       1 UNK1        -0.0533
     21 C5S          7.7880    6.8910   -0.2780 C.3       1 UNK1        -0.0559
     22 C6S          8.5610    6.0120    0.6940 C.3       1 UNK1        -0.0653
     23 O11          2.0210    6.8650   -6.5580 O.co2     1 UNK1        -0.6464
     24 O12          1.5600    8.9920   -7.1940 O.co2     1 UNK1        -0.6464
     25 O31         -0.8270   16.3320    1.5360 O.co2     1 UNK1        -0.6469
     26 O32         -1.2220   14.1290    1.3280 O.co2     1 UNK1        -0.6469
     27 HA1         -0.6754    8.6322   -5.9842 H         1 UNK1         0.0710
     28 HA2          1.5795   11.2752   -0.3036 H         1 UNK1         0.0565
```

```
     29 HA31        1.6855   15.6383    1.1583 H         1 UNK1        0.0539
     30 HA32        0.8841   15.6643   -0.4471 H         1 UNK1        0.0539
     31 HB11        1.1089    7.5522   -3.6916 H         1 UNK1        0.0295
     32 HB12       -0.5840    8.1441   -3.5645 H         1 UNK1        0.0295
     33 HB21        3.8787   12.2592   -0.1452 H         1 UNK1        0.0396
     34 HB22        3.8644   12.5463   -1.9154 H         1 UNK1        0.0396
     35 HG11        0.1080   10.3813   -4.5640 H         1 UNK1        0.0356
     36 HG12        1.8015    9.7823   -4.5302 H         1 UNK1        0.0356
     37 H1S1        2.5670    9.4584    0.1645 H         1 UNK1        0.0376
     38 H1S2        4.1417    9.8745    0.9215 H         1 UNK1        0.0376
     39 H11        -0.7693    6.3340   -6.6269 H         1 UNK1        0.2751
     40 H12        -1.3933    6.5155   -5.0647 H         1 UNK1        0.2751
     41 H13         0.1120    5.8967   -5.3077 H         1 UNK1        0.2751
     42 H2S1        3.9108    7.3839   -0.8799 H         1 UNK1        0.0273
     43 H2S2        3.5014    7.3929    0.8713 H         1 UNK1        0.0273
     44 H2          1.3801   12.0166   -3.1799 H         1 UNK1        0.1311
     45 H3S1        5.6560    7.7320    1.6045 H         1 UNK1        0.0266
     46 H3S2        6.0582    8.7446    0.1770 H         1 UNK1        0.0266
     47 H3          1.6105   13.1450    0.9963 H         1 UNK1        0.1310
     48 H4S1        5.8445    6.4428   -1.1939 H         1 UNK1        0.0265
     49 H4S2        6.1244    5.7420    0.4362 H         1 UNK1        0.0265
     50 H5S1        8.0365    7.9685   -0.1352 H         1 UNK1        0.0263
     51 H5S2        8.1089    6.6095   -1.3081 H         1 UNK1        0.0263
     52 H6S1        9.6604    6.1862    0.6291 H         1 UNK1        0.0230
     53 H6S2        8.3125    4.9345    0.5512 H         1 UNK1        0.0230
     54 H6S3        8.2401    6.2935    1.7241 H         1 UNK1        0.0230
@<TRIPOS>BOND
     1    1    4 1
     2    1   10 1
     3    1   12 1
     4    1   27 1
     5    2    5 1
     6    2   13 1
     7    2   15 1
     8    2   28 1
     9    3   17 1
    10    3   19 1
    11    3   29 1
    12    3   30 1
    13    4    6 1
    14    4   31 1
    15    4   32 1
    16    5    7 1
    17    5   33 1
    18    5   34 1
```

```
     19     6     8 1
     20     6    35 1
     21     6    36 1
     22     7    11 1
     23     8     9 2
     24     8    15 am
     25    10    23 ar
     26    10    24 ar
     27    11    14 1
     28    11    37 1
     29    11    38 1
     30    12    39 1
     31    12    40 1
     32    12    41 1
     33    13    16 2
     34    13    19 am
     35    14    18 1
     36    14    42 1
     37    14    43 1
     38    15    44 1
     39    17    25 ar
     40    17    26 ar
     41    18    20 1
     42    18    45 1
     43    18    46 1
     44    19    47 1
     45    20    21 1
     46    20    48 1
     47    20    49 1
     48    21    22 1
     49    21    50 1
     50    21    51 1
     51    22    52 1
     52    22    53 1
     53    22    54 1
@<TRIPOS>SUBSTRUCTURE
      1 UNK1        1 RESIDUE           1 A     UNK     0 ROOT
```