DockControl: a New Integrated Software for Design of Experiments and Molecular Docking: Application to HIV-Protease Inhibitors

Eva Nylander

December 18, 2007
Master's Thesis in Computing Science, 20 credits
Supervisor at Department of Chemistry, Umeå University: Anna Linusson
Supervisor at CS-UmU: Per Lindström
Examiner: Lennart Edblom

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

In this thesis the development of a software is described. The software integrates a number of scripts and docking programs with the purpose of facilitating research concerning the influence of different docking parameters on docking performance. The software is tried on a number of HIV-proteases with a good result. The thesis also contains an overview of different sampling function and a comparison between a number of docking programs.

DockControl: a New Integrated Software for Design of Experiments and Molecular Docking: Application to HIV-Protease Inhibitors.

Sammanfattning

Rapporten beskriver arbetet med att ta fram en mjukvara som sammanför ett antal skript och dockningsprogram till en enhet. Mjukvaran kommer i framtiden att användas för att underlätta forskning där man undersöker effekten av olika parametrar på resultatet av dockningar mellan proteiner och ligander. I rapporten diskuteras också olika sampling funktioner och jämförelser görs mellan ett antal dockningsprogram.

Contents

1 Abbreviations

GOLD	Genetic Optimization for Ligand Docking
FRED	Fast Rigid Exhaustive Docking
MOE	Molecular Operating Environment
RIPS	Random Incremental Pulse Search
RMSD	Root-Mean-Square-Distance
GA	Genetic Algorithm
LGA	Lamarckian Genetic Algorithm
MWT	Molecular Weight
PLS	Partial Least Square regression to latent structures
PDB	Protein Data Bank

2. Introduction

In this thesis the implementation of a software simplifying the work flow of a method for optimizing parameters for docking programs is described, the problem description can be found in page 17. This thesis also includes two in-depth studies, one comparing sampling algorithms for docking programs and one describing drug-protein binding, these are found in the following section.

3 Background

An important branch in chemistry is the development of drugs, drug design. To design a drug is a complex and time-consuming procedure, which consists of three phases, each of which can be divided further. The phases are drug discovery, drug design and, finally, drug testing and development [1]. Over the years software has gotten a more pronounced role in this procedure and today software are commonly used by most drug companies.

One specific group of software used in drug design is the docking programs. These programs model the binding of small molecules (ligands) to bigger molecules with a biological importance (e.g. Proteins). There are three main uses for docking programs; prediction of the binding mode of a known binding ligand, identification of new ligands using virtual screening and prediction of the binding affinities of related compounds from a known active series [2]. Prediction of the binding mode of a known active ligand is the most basic assignment of a docking program, it is this category that the other two relies on and it is also the category in which most success has been achieved. A virtual screening aims at identifying new lead compounds (new templates for drugs) in huge data bases consisting of hundreds of thousands of molecules. Each of these molecules must be docked into the protein receptor of interest, the complementarity of the two molecules must be calculated and, finally, the molecule must be either collected or discarded based on how well it interacts with the receptor. Docking programs are today capable of distinguishing between unlikely and likely candidates, but the programs are unable to rank the latter [2]. The last of the mentioned categories are by far the most complicated and useful one, unfortunately, there exist no program which is capable of ranking ligand affinity accurately regardless of receptor and binding types. However, there do exist programs which are capable of ranking ligands for a particular protein target or for a particular family of targets.

All docking programs have the same basic functions even though the implementation of these functions can vary. A docking program generally solves the docking through two steps. In the first step the conformational space (i.e. the space of the different orientations of the ligand produced by rotating about bonds) of the ligand is explored in search for an energy minima, corresponding to a conformation of the ligand which has lower energy than the conformations surrounding it. The function searching this space is called the sampling function. In the second step the found poses are ranked according to how well the ligand and the target fit together, this function is called the scoring function. The available sampling functions can be divided in three categories: systematic, random (or stochastic), and simulation methods [3]. The type of scoring function can also be divided in three categories: force-field-based, empirical and knowledge-based. Scoring functions try to measure biological activity based on the bonds and interactions that occur between the ligand and the target. Since there are so many interactions and the scoring functions all values varying interactions differently, separate scoring functions often perform best on specific groups of targets and ligands.

Docking programs performs the docking based on representations of the protein templates and their ligands. These representations describe the coordinates of the molecules in 3D. The

coordinates are deduced from crystals examined using spectroscopy. Most programs treat the ligand as flexible and the protein receptor as rigid or nearly rigid.

3.1 Comparison of docking programs

As a part of this thesis a number of docking software were discussed based on their sampling functions. The software discussed were three software which the department was licensed for: these are GOLD (Genetic Optimization for Ligand Docking) [4, 5], FRED (Fast Rigid Exhaustive Docking) [6], and MOE (Molecular Operating Environment) [7] together with the most cited docking software today AutoDock [8, 9]. The discussion will be focused on the computer science effect of the sampling functions. In this way discussion about the chemical properties of ligands that are suited to be or not to be docked by the software in question is avoided. The mentioned software have different sampling functions, one of the software has a function based on a mix of tabu and Monte Carlo search (MOE), one applies a systematic search using depth-first (FRED) and two uses differing variants of genetic algorithms (GOLD and AutoDock). In the subsequent sections the software will be discussed one by one, a comparison of the performance of the software will be made based on scientifically published papers and this comparison will be discussed in the light of the different properties of the sampling functions.

3.1.1 The implementation of sampling functions

MOE

MOE actually uses two kinds of sampling functions, one systematic, used for smaller molecules, and one stochastic, for bigger molecules. Here, the discussion will focus on the stochastic search. The search is based upon random rotations of ligand bonds. Upon rotation the afforded new ligand conformation is energy minimized (to accomplish a stable conformation), to avoid being stuck on saddle points, the coordinates of each atom is perturbed, and again the energy is minimized. The resulting conformation is compared to a conformer list, if there exist no duplicate in the list, the new conformation is added to the list and the search continues with the new conformation as template. The method is based on the Random Incremental Pulse Search method (RIPS) [10], a method described as a Monte Carlo type routine employed to randomly search the potential energy surface for a given ligand.

FRED

The actual sampling of ligand conformations does not occur in FRED. The program instead takes a file describing the interesting conformations of the ligands and docks them. The conformation file usually is constructed in a software called OMEGA [11], supplied by the same vendor as FRED.

OMEGA constructs conformational ensembles by dividing the ligand into fragments, separated by rotatable bonds. Each bond is then given a set of possible dihedral angles, and then groupings are made, each containing up to five contiguous bonds. Exhaustive depth first torsion search is then performed on each of these fragments, and the resulting fragment conformers are then sorted according to energy. Entire structures are assembled by combining the lowest energy set of fragments, then the next lowest until the search is terminated. A final ensemble is selected where each conformer must have a root-mean-square-distance (RMSD) to every other member of the ensemble that exceeds a cut off value. This method generates conformers extremely rapidly.

GOLD

GOLD uses a sampling algorithm that allows full acyclic ligand flexibility, partial cyclic ligand flexibility and partial protein flexibility in the neighbourhood of the active site. The

algorithm is based on a steady-state operator-based genetic algorithm (GA) which implies that the sampling is non-deterministic, time-consuming and that the result is not completely reproducible. The algorithm is as follows:

1. A set of reproduction operators (crossover, mutation etc.) is chosen. Each operator is assigned a weight.
2. An initial population is randomly created and the fitness of its members determined.
3. An operator is chosen using roulette wheel selection based on scaled fitness.
4. The parents required by the operator are chosen using roulette wheel selection based on scaled fitness.
5. The operator is applies and child chromosomes produces. Their fitness is evaluated.
6. If not already present in the population, the children replace the least fit members of the population, i.e. the members with the lowest fitness score.
7. If 100 000 operators has been applied stop, otherwise go to 3.

The chromosomes contain information about both the ligand and parts of the protein. The information is encoded by two binary and two integer strings, one each for the ligand and the protein. The binary strings encode the angles of rotation about rotatable bonds, and the integer string suggests possible hydrogen bonds between the ligand and the protein.

Three operators are used in this GA: mutation, crossover and migration, and the operator is chosen using a roulette-wheel selection based on operator weights. The crossover operator performs two-point crossover on integer strings and one-point crossover on binary strings. The mutation operator performs integer-mutation on the integer strings and binary mutation on the binary strings. The proceeding of the mutation operator is made through copying the parent chromosome to the child chromosome and thereafter randomly performing mutation on one of the four strings. The last of the operators is justified by the fact that the algorithm uses a distributed environment, known as the island model, involving several sub-populations. As implied by its name, the migration operator copies an individual from one island to a neighbouring island. Upon the migration of one individual a technique known as nicheing is employed to further increase the population diversity. This involves comparing the new individual against all other individuals in the population and based upon this determine if any other individual was found in the same niche as the new individual. If another chromosome is found in the same niche the new chromosome replace the least-fit member of the niche rather than the least-fit member of the population.

AutoDock

AutoDock uses a hybrid sampling function between a GA and a local search function, the hybrid is called a Lamarckian genetic algorithm (LGA).

What differentiates the LGA from a regular GA is that the LGA employs a local search at the end of each generation. The local search is made in the genotypic space instead of the more typical phenotypic space and the “environmental adaptations” acquired by the individual during the search will be inherited by its offspring.

The overall algorithm looks as follows:

1. Create a random population of individuals.
2. Mapping and fitness evaluation.
   In this step each individual genotype is translated to its corresponding phenotype and thereafter evaluated according to its fitness.
3. Selection.
   The number of offspring from each individual is decided. The number is based on the

fitness of the individual.

4. Crossover and mutation.
   Crossover and mutation, in that order, is performed on random members of the population.
5. Elitist Selection.
   The new population is evaluated according to its fitness and a number of the highest ranked individuals survive into the next generation.
6. Local search.
   The search is performed on previously defined portion of the population.
7. If the maximum number of generations or the maximum number of energy evaluations is reached stop, otherwise go to 2.

In this implementation the chromosome is composed of a string of real-valued genes: three Cartesian coordinates for the ligand translation; four variables defining a quaternion specifying the ligand orientation; and one real-value for each ligand torsion. When creating the first population a specific hardware-independent random number generator is used, which allows result to be reproduced on any platform given the same seed values.

The crossover operator is implemented using two-point crossover, with breaks only occurring between genes. The offspring replaces the parent in the population. The mutation operator is performed by adding a random real number that has a Cauchy distribution to the variable. The Cauchy distribution has thick tails that enables it to generate large changes occasionally.

The local search method is based upon a method developed by Solis and Wets [12], which does not require gradient information about the energy landscape. The local search method is also adaptive in that it adjusts the step size depending upon the recent history of energies.

3.1.2 Comparison of sampling functions

A substantial number of articles discussing the performance of different docking program are available. Four articles have been selected as foundation for the comparison between FRED, MOE, GOLD and AutoDock, unfortunately none of the articles features all docking programs, instead the articles compares AutoDock and MOE [13], AutoDock and GOLD [14], FRED and GOLD [15] and, finally, FRED, GOLD and MOE [16]. Unfortunately, no paper relating the performances of AutoDock and FRED could be found.

The first of the articles compares AutoDock and MOE to a new method for docking. The essence of the AutoDock-MOE comparison [13] is:

1. MOE is much more computationally expensive than AutoDock (average calculation time in minutes per structure is 5.15 for AutoDock and 37 for MOE)
2. The differences between the ability to reproduce the ligand X-ray structures are quite small. However, MOE comes across as slightly more accurate than AutoDock.

The main conclusions drawn from the comparison between AutoDock and GOLD [14] were:

1. Both AutoDock and GOLD docks 46% of ligands from a cross-docking experiment correctly within 2 Å RMSD of their known binding mode.
2. The fraction of molecules that are docked with acceptable accuracy is 0.47 for AutoDock and 0.52 for GOLD, this accuracy is considered as approximately the same in the article.
3. The average docking times is consistently higher for AutoDock than for GOLD; for some receptors even more than twice as high.

When comparing FRED and GOLD the following results were made [15]:

1. At a 1 Å RMSD cut-off, docking was successful for 61-63% of the cases using GOLD, while for FRED, the success rate was 29-38%.
2. GOLD placed about 80-90% of the ligands of the data set within 2.0 Å of the X-ray pose, while FRED was considered successful in 62% of the cases.
3. For GOLD the docking times in this study ranges between 55-479 seconds, with a mean of 137 seconds. For FRED the range is 0.1-193 seconds, with a mean of 18 seconds.
4. GOLD works well for small binding sites, while FRED achieves best performance for medium-sized binding sites. Both GOLD and FRED also works well for large protein cavities.
5. The docking accuracy is inversely proportional to the conformational freedom of the ligand for GOLD; this effect is much less pronounced for FRED.

When comparing FRED, GOLD and MOE [16], lastly, it can be concluded that:

1. In general, Gold performs better than FRED, which performs better than MOE.
2. MOE performs generally bad.

3.1.3 Analysis

The articles give somewhat contradictory information. Tøndel et al. claims that MOE is slightly more accurate than AutoDock, at the same time the study published by Bursulaya et al. arrived at the conclusion that AutoDock is almost equal to GOLD, which in turn outperforms MOE in Warren et al. The different experiences by the authors are hard to explain, but it is a common knowledge that optimal settings for parameters in a program differs for different target-ligand pairs and non-optimal settings can affect the docking performance profoundly [17]. Also, the nature of the protein datasets used in the different studies can influence the outcome. It is believed that different programs are suitable to use for specific types of proteins, hence the data set can be more suited for a specific docking program in one study and not in another. To be able to reliably compare the programs the contradictions were discussed with researchers experienced in using FRED, GOLD and MOE for docking. Their unanimous opinion was that MOE performs consistently worse than GOLD and FRED much in accordance with the last article. As stated above no article could be found relating the performance of FRED and AutoDock, but as Kellenberger et al. has reached the conclusion that FRED performs worse than GOLD, while Bursulaya et al. claims that AutoDock is almost equal to GOLD it seems as a safe assumption to make that AutoDock performs better than FRED. The assumption is further prompted by analyzing the diagrams in the article by Warren et al., although no quantitative numbers is accounted for. In summary, the program performance increase in the order MOE-FRED-AutoDock-GOLD, and the docking times increases in the order FRED-GOLD-AutoDock-MOE.

What causes the different performances and the docking times for the four programs? Even though the programs contain both sampling and scoring of different poses the sampling is the most time-consuming part, the time requirement of the program can be approximated by the time-requirement of the sampling functions. The fastest docking program by far is FRED, which is the only program that encompasses a systematic search. The time statistics stated above should perhaps be considered with care since the sampling takes place in OMEGA, but when the time requirements for FRED and OMEGA together is taken into account the speed

is still regarded as very fast compared to other docking programs. But how is it possible that a program using systematic search is the fastest? No explanation can be found hidden in the documentation of OMEGA, but in Leach [18] a plausible explanation can be found. It is stated that many programs using a systematic search eliminate structures having a very high energy or some other problem, from the time-consuming energy minimization stage. It is also possible to further enhance the efficiency of the search by checking partially constructed conformations, if they do not fulfil the requirements regarding stability and so on, they are discarded. The programs compared with OMEGA are uses stochastic searches, and these kinds of searches are renowned for taking much time to complete. Both AutoDock and GOLD uses a GA as sampling function, in the article comparing these programs it is stated that the average docking time is higher for AutoDock than for GOLD. The probable cause for the longer docking time of AutoDock is the local search performed in each step. The most time-consuming program in this comparison is MOE, a possible reason for this is that very small atomic displacements are required to achieve an acceptable acceptance ratio in Monte Carlo simulations, this means that the phase space is covered very slowly [18].

AutoDock and GOLD is the programs that perform best in terms of finding the most probable docking pose of the ligand, the programs are even stated to be comparable in performance. Both AutoDock and GOLD are based on a GAs but takes different approaches to enhance it: AutoDock uses a local search and GOLD uses the island model, a model developed to maintain diversity without hindering the optimization efficiency. FRED performs much better than MOE, perhaps this depends on the very slow covering of the phase space by Monte Carlo simulations, resulting in the possibility that the program has not been given enough time to cover the search space. It could even be possible that the search space for the used molecules is of such a magnitude that the time needed to cover it is unrealistic. The GAs in AutoDock and GOLD perform consistently better than the systematic search in FRED. This is strange because a systematic search should by definition find all conformations; if all conformations are found then the most fitting conformations would also be found. But why then does not OMEGA find the conformations the GA find? OMEGA generates the conformations with the lowest energy, but the correct docking conformations are not necessarily the ones with the lowest energy. A GA also searches for optimums, but since GAs also can produce solutions very close to the global optimum [18], it is possible that the GA utilized by GOLD find the correct conformations with not so low energy, where OMEGA find the conformations that is not precisely correct but with the lowest energy.

In this section some aspects of the sampling algorithms for different docking programs have been accounted for. The sampling algorithms are, however, not the only thing affecting the performance of a docking program. The scoring functions also has a profound role, and, furthermore, the scoring functions also affect how well suited the programs are in handling different kinds of molecules. This is also an aspect that needs to be considered when choosing a docking program.

3.2 Interactions between proteins and drugs

The interaction between a drug and a protein is quite complex and makes high demands on the geometry of functional groups for making different kinds of bonding possible, hence the problems with accurate scoring in docking algorithms (see above). Nonetheless it is possible to distinguish different requirement, that when fulfilled, makes the function of a molecule as a drug more probable. In an article Lipinski et al [19] has formulated ‘the rule of 5’, this set of rules states that when a number of requirements are not fulfilled poor absorption or permeation of drug molecules is more likely. The rule is stated as follows:

Poor absorption or permeation is more likely when:

There are more than 5 H-bond donors (expressed as the sum of OHs and NHs)
The MWT (the molecular weight) is over 500
The Log P is over 5 (or MLogP is over 4.15)
There are more than 10 H-bond acceptors (expressed as the sum of Ns and Os)

Compound classes that are substrates for biological transporters are exceptions to the rule.

The Log P mentioned above is the lipophilicity of the molecule expressed as the logarithm of the ratio of octanol solubility to aqueous solubility. There are a number of ways to calculate this value but the two mentioned in the article are CLogP (equal to Log P in the rules above) and MLogP, where CLogP is recommended by Lipinski et al, but if no such value can be acquired for the molecule in question MLogP can be calculated. CLogP is usually calculated by software based on the article of Leo et al [20], while MLogP is calculated according to the formula suggested by Moriguchi et al [21, 22]:

$$\log P = 1.244(CX)^{0.6} - 1.017(NO)^{0.9} + 0.406PRX - 0.145(UB)^{0.8} + 0.511HB + 0.268POL -$$
$$2.215AMP + 0.912ALK - 0.392RNG - 3.684QN + 0.474NO2 + 1.582NCS + 0.773BLM - 1.041$$

Where the meaning of the different parameters is as stated in table 1.

The reason for excluding classes that are substrates for biological transporters is that these classes get a natural way to their protein by the biological transporters they bind to. The classes are antibiotics, antifungals, vitamins and cardiac glycosides.

The rules as stated above were derived from examinations of a library of compounds having entered Phase II efficacy studies. It could be interesting to note that when combining any two of the four parameters outside the desirable range, none of these combinations exceeded 10%. Furthermore, the combination of high MWT and high LogP has such a low percentage as 1. Lipinski et al now uses the rule as an alert system when registering new compounds and in their article it is stressed that even though a compound that not fulfil 'the rule of 5' will likely have a poor bio-availability, the converse is not true, meaning that compounds fulfilling the rules can still be troublesome in experimental studies.

Parameter	Description
CX	Summation of numbers of carbon and halogen atoms weighted by C: 1.0, F: 0.5, Cl: 1.0, Br: 1.5, and I: 2.0
NO	Total number of N and O atoms.
PRX	Proximity effect of N/O; X-Y: 2.0. X-A-Y: 1.0 (Y, Y: N/O, A: C, S, or P) with a correction (-1) for carboxamide/sulphonamide
UB	Total number of unsaturated bonds except those in NO_2
HB	Dummy variable for the presence of intramolecular hydrogen bond as *ortho*-OH and –CO-R, -OH and $–NH_2$ and –COOH, or 8-OH/NH_2 in quinoxalines, etc.
POL	Number of aromatic polar substituents (aromatic substituents excluding Ar-CX_2- and Ar-CX=C<, X: C or H)
AMP	Amphoteric property; α-aminoacid: 1.0, aminobenzoic acid: 0.5, pyridinecarboxylic acid: 0.5
ALK	Dummy variable for alkane, alkene, cycloalkane, or cycloalkene (hydrocarbons with 0 or 1 double bond)
RNG	Dummy variable for the presence of ring structures except benzene and its condensed rings (aromatic, heteroaromatic, and gydrocarbon rings)
QN	Quarternary hydrogen: $>N^+<$, 1.0; N oxide, 0.5

NO2	Number of nitro groups
NCS	Istohiocyanato (-N=C=S), 1.0; thiocyanato (-S-C≡N), 0.5
BLM	Dummy variable for the presence of β-lactam

Table 1. The parameter description for the parameters in the equation for calculating MLogP.

For a drug to bind to its target there are a number of requirements on the protein cavity also, these are accounted for in an article by Nayal et al [23]. In the article the group has analyzed the important properties of the protein cavities to which drugs bind. Nine attributes yielding a high-confidence prediction of drug binding have been found, interestingly, all of these attributes describe cavity shape, whereas none describe physicochemical properties. The seven attributes are

A number of contributing residues between 32 and 54
A number of contributing atoms between 120 and 261
An average depth between 6.8 Å and 11.4 Å
A maximum depth between 13 Å and 22.9 Å
A deviation of surface vertices depth distribution between 3.0 Å and 5.6 Å
A largest moment of inertia between 10^5 and $2.2x10^5$
A minimum moment of inertia between $3.8x10^4$ and 10^5, or a normalized minimum moment of inertia between 26.6 and 48.7.

The cavity rank for a surface cavity is achieved by comparing all cavities of the protein; the largest cavity gets rank 1 and so on.

While the mentioned rules for drugs and protein cavity gives a good appreciation of what properties a good drug and its target should have, the docking programs used in laboratories need to make an appreciation of the affinity between specific drugs and proteins. These appreciations are made based on more mathematical and physical grounds.

The non-covalent, reversible association between a ligand and a protein generally occurs in an aqueous, electrolyte-containing solution, this reaction is determined by the standard Gibb's free energy of binding (ΔG°), also called (binding) affinity, which quantifies the tendency of a molecule to form a complex together with another one according to:

$$\Delta G° = \Delta H° - T\Delta S°$$

Where T refers to the absolute temperature and ΔH° and ΔS°, refers to the enthalpic and entropic contributions of the equation. The non-covalent binding is determined by several parts, these are electrostatic interactions, solvation and desolvation contributions, van der Waals interactions, and intramolecular changes of receptor and ligand during complex formation [24].

There are several kinds of electrostatic interactions, among those the hydrogen bonding is the most prominent one. A hydrogen bond is an electrostatic attraction between a hydrogen atom bound to an electronegative atom X and an additional electronegative atom Y or a π–electron system. Distance of 2.5-3.2 Å between hydrogen-bond donor X and acceptor Y and X-H···Y angles of 130-180° are typically found. The strength of a hydrogen bond depends on its environment and more precisely on the local dielectric constant ε of its surrounding medium; the constant affects the shielding of electrostatic interactions. The interaction energy of a hydrogen bond is proportional to ε^{-1}, and since the dielectric constant is higher in the protein interior and in close proximity to polar groups buried hydrogen bonds is regarded as more important for protein-ligand interactions than those formed in solvent exposed regions. Evidence from protein-mutation studies suggest values for the interaction between uncharged

partners of $\Delta G°=-5\pm2.5 kJmol^{-1}$, in contrast, values of -10 to -20 $kJmol^{-1}$ have been reported for charge-assisted hydrogen bonds [24].

In aqueous environment water molecules form a tetrahedrically connected network of hydrogen bonds, both between two water molecules, and between a water molecule and the protein and/or ligand. For the ligand and protein to form a complex the already existing hydrogen bonds and formed network must be broken and new hydrogen bonds between the water molecules and the newly formed complex must be made, this process is called the desolvation (of the separate ligand and protein), solvation (of the newly formed complex) process. The process also involves the reorganization of the water structure at the complex interface. The behaviour of water forming network of three to four hydrogen bonds per molecule can also be found among interstitial water molecules mediating interactions between protein and ligand.

Upon complex formation both the ligand and the protein experience changes in degrees of freedom, resulting in a change of entropy. Even if it is tempting to conclude that the molecules only experiences a restricted mobility, experimental evidence exist indicating that ligands frequently retain considerable mobility even in bound state and a protein can even experience an increase in mobility.

When trying to find a new drug, scientists need the docking programs to approximate the affinity between the possible new drugs and the protein of interest; this is the aim of scoring functions. Since the calculation of the standard Gibb's free energy of binding is expensive and remain impractical for the evaluation of large numbers of protein-ligand complexes, scoring functions implemented in programs make various assumptions and simplifications in the evaluation of modelled complexes. The scoring functions can, according to their approach to this question be divided into three classes: force-field-based, empirical and knowledge-based [3].

The force-field based scoring functions usually calculate two energies, the receptor-ligand interaction energy and the internal ligand energy. Most force-field based scoring function only considers a single protein conformation, and this makes it possible to ignore the internal protein energy. The receptor-ligand interaction energy and the internal ligand energy are mostly described by calculating van der Waals and electrostatic energy terms. The van der Waals energy term is calculated by using a Lennard-Jones potential function:

$$E_{vdW}(r) = \sum_{j=1}^{N}\sum_{i=1}^{N} 4\varepsilon\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

The shown equation is a Lennard-Jones 12-6 function where ε is the well depth of the potential, σ is the collision diameter of the atoms i and j, and r is the distance between i and j. The exponents 12 and 6 are responsible for the small-distance repulsion and the attractive forces between the atoms, respectively. High exponents results in increasingly repulsive potentials and is less forgiving of close contacts between atoms, to use lower terms like in the 8-4 Lennard-Jones potential makes the potential softer.

Electrostatic terms are calculated by pair-wise summation of Coulombic interactions:

$$E_{coul}(r) = \sum_{i=1}^{N_A}\sum_{j=1}^{N_B} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}$$

Here, N is the number of atoms in molecules A and B, respectively, q is the charge on each atom, ε_0 is the vacuum permittivity and r is the distance between atoms i and j. The two

latter equations illustrates how the van der Waals potential and electrostatic energy terms are calculated between the ligand and the protein, the energy terms for the internal ligand energy are calculated in a similar manner. Force field scoring functions has two major limitations, the biggest being that they where originally formulated to model enthalpic gas-phase contributions to structure and energetic and therefore takes no consideration to solvation and entropic terms. The second limitation is that the calculations require cut off distances for the treatment of non-bonded interactions, these are mostly arbitrarily chosen and complicate the handling of long-range effects in binding.

The empirical scoring functions are based on the idea that binding energies can be approximated as a sum of uncorrelated terms. The coefficient of each terms are calculated by regression analysis based on experimentally determined binding energies. The resulting formulas are often simpler than the force-field scoring functions even though many of the individual terms have counterparts in force-field molecular dynamics terms.

The advantage of the empirical functions is that they often are simple to evaluate, the disadvantage is that the functions are dependent on the experimental data used for the regression analysis; as a consequence of this differently fitted empirical scoring functions can not easily be combined into a new function [3].

The knowledge-based scoring functions are designed to reproduce experimental structures rather than binding energies. The complexes are modelled using relatively simple atomic interaction-pair potential, defined on their molecular environment. These kind of scoring functions are computationally simple, and can therefore be used in screening of large databases, but they are derived from information implicitly encoded in limited sets of protein-ligand complex structures [3].

4 Goals and purposes

In most docking programs the docking is controlled by a diverse set of parameters. These parameters control virtually every aspect of docking and can influence the docking results in a very profound way. At the chemistry department at Umeå university research is made to investigate the influence of different docking parameters with the aim to enhance docking performance [17]. The software used in this study were GOLD and FRED. The investigations are made using a specific project methodology, depicted in figure 1. To find the parameters with the most influence a number of files (called design, ligand and target files in the figure) are generated and offered to the script controlling the docking program (docking script). The script executes the docking runs defined in the design file and the output is then given to the RMSD script. This script calculates the RMSD between each ligand conformation generated by the docking program and the experimentally determined ligand conformation. The interesting results (the results ranked the highest by the docking program and the results with the lowest RMSD) is extracted using the top and best pose scripts, and the extracted values are then used to construct models explaining the relations between the RMSD values and the values of the input parameters. The models are constructed using partial least square regression to latent structures (PLS).

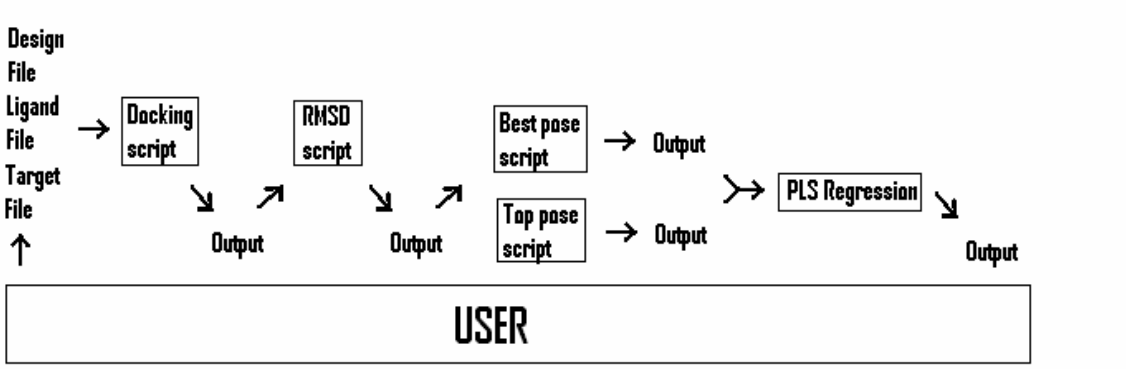


Figure 1. The project methodology for investigate the influence of different docking parameters on the performance of docking programs.

The above described methodology is based on a number of program and several scripts, each handled and modified by the user before each run, making the procedure more complicated than needed. The aim of this thesis is to provide a tool which combines the different steps in the procedure. The requirements on the tool is that is should be based on existing scripts, easy to use, robust, flexible with a possibility to add new modules and well documented.

4.1 The ordinary procedure

The application aims at simplifying the investigation of the influence of specific parameters on docking performance. The procedure that will be simplified looks like this:

1. A number of complexes (ligand + target protein) are identified and prepared for a docking experiment.
2. The docking parameters of interest are identified and the settings for them are decided.
3. A statistical design of the parameters and their settings are made.
4. If necessary, a file describing the active site of the target and/or a file describing the different conformations of the ligand are manufactured.
5. The docking programs are run, using Perl scripts controlling them and managing the parameter settings.
6. The RMSD between the experimentally determined ligand conformation and the different conformations resulting from the docking simulation are calculated using a Perl script.
7. The top pose and best pose resulting from the run is extracted using two Perl scripts.
8. The median RMSD for top pose and best pose across all ligands are extracted.
9. The relationships between the different parameter combinations in the statistical design and the median RMSD for top and best pose are evaluated using PLS regression.

4.2 The role of the application

The application should contain some of the steps described above; the resulting procedure should look like this:

1. A number of complexes (ligand + target protein) are identified and prepared for a docking experiment.
2. If necessary, a file describing the different conformations of the ligand is manufactured.

3. The application is started
    3.1 If necessary, a file describing the active site of the target is manufactured.
    3.2 The parameters of interest are identified and the settings for them are decided.
    3.3 A statistical design of the parameters and their settings is made.
    3.4 The post processing of the output data is chosen.
    3.5 The application executes the wanted docking program and the scripts controlling the post processing of data.
4. The relationship between the different parameter combinations in the statistical design and the median RMSD for top and best pose are evaluated using PLS regression.

4.3 The requirements

Further requirements on the application are as follows:

- **The application should be based on existing scripts.** The docking programs are controlled by and the post processing of data is done by the use of Perl scripts. The incorporation of the already existing scripts facilitates the possibility for a user to directly affect the program run when using the application without the need of actually managing the code of the application. The department has personnel skilled in Perl programming and considerations are made concerning further use of Python as script language, therefore it would be preferable if the application could control scripts written in both languages.
- **The application should make the work flow smoother and make it possible to save time.**
- **The application should be easy to use.**
- **The application should be robust.**
- **The application should be flexible.** Today two docking programs are used at the department, and purchases of more are planned in the future. Since the application is thought to be a common tool in the department, it should be possible to run any docking program using it. The possibility to only use parts of the application independently of the other features should exist as well.
- **The application should be well documented.** The research environment has many collaborators and there is a possibility for the application to be transferred between different groups and departments. Since research methodology changes rapidly and end users always have new requests of an existing program the application should be well documented, to facilitate future modifications and changes of the application.
- **The application should be able to use on different OS's.** This requirement is a consequence of the previous one.
- **The application should preferentially have a graphical user interface (GUI).**

4.4 The idea

The current procedure, especially the manifold scripts and their location, mean a lot of work for the researcher wanting to do a docking. In figure 1 the work load for the user for each docking is depicted. For each docking run, a number of files need to be supplied: the files are describing the target and the ligands and also the design used in this specific run. The information of whereabouts of these files is manually written into the docking scripts, and thereafter the script is run. The output from the docking script is moved to a location on another computer where the RMSD script resides, also for this script information are manually inscribed in the script and then the script is executed. The same procedure (e.g. manually entering information about the whereabouts of the files resulting from the preceding

run and then executing the scripts) is performed for the top pose and best pose scripts. Finally the outputs from these scripts are used to construct a PLS regression using SIMCA-P+ [25].

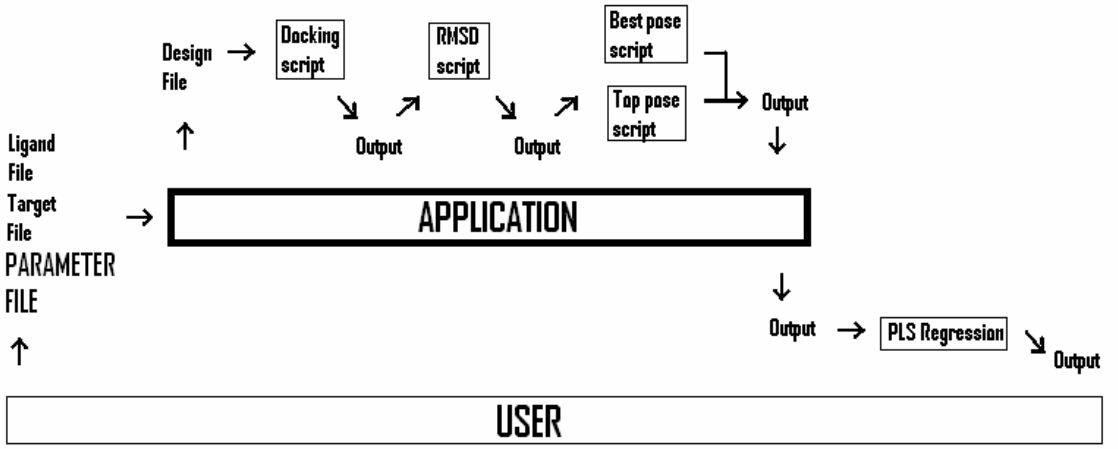


Figure 2. The figure describes the role of the application in the project methodology. In comparison with figure 1, the application now manages the transfer of information between scripts. The application also handles the construction of the design file, supplied to the docking script.

To simplify the work for the user the application will be used as visualized in figure 2. The role of the application is to relieve the user of some of the work. The idea is that the application receives the whereabouts of the information on the ligand and the target, the settings for the design file, and how the output from the docking program will be handled. Thereafter, the application constructs the design file, executes the different scripts, and delivers the resulting output to the user. During the development of the application some modifications of this idea was made resulting in a new file, the parameter file (written with capital letters in the figure), given as argument to the application. The parameter file will be accounted for in section 3.4.

5 The preparations

5.1 The language

The language in which to write the application was decided to be Java much because this language is well suited for writing a portable application. The language also has an extensive library for building a graphical user interface, which is an attractive characteristic. Furthermore, it is quite easy to handle communication between Java and Perl and also between Java and Python. The code was written in Fedora Eclipse [26] using Java 1.5.0 [27].

5.2 The scripts

As stated in the requirements the application should be flexible, meaning that the application should be able to control different docking programs and different scripts. In the original procedure six Perl scripts was used

1. A script constructing a file describing the active site for a given protein.
2. A script controlling and running the docking program FRED
3. A script controlling and running the docking program GOLD
4. A script calculating the RMSD between the output conformations and the experimentally determined conformation of the ligand

5. A script extracting the best pose
6. A script extracting the top pose

Script number 2 and 3 should be interchangeable in the application. The remaining scripts can either be executed or not executed at different runs.

5.2.1 The scripts controlling docking programs

The scripts controlling the two available docking programs were at the start of this project handled differently. The script controlling FRED took no input, instead the user manually changed three variables inside the script. These were paths to the directory in which the output would be saved, the directory containing information about the ligand/s, and the directory containing information about the target/s.

The script controlling GOLD took one input, a name of a file containing all protein database (PDB) codes for the complexes that would be docked. The script controlling FRED has no corresponding variable. The user also manually changed four variables in the script for every run; these variables are the paths to the home directory of the run, the directory containing information about the ligand/s, the directory containing information about the protein/s, and the directory in which the output will be saved.

Besides the already mentioned variables both scripts had variables for the name of the file containing the design for the current run, the name of a file containing all variables used for running the specific docking program and a string denoting the command issued whenever a run of the specific docking program will be performed.

The two docking scripts were rewritten so they would have an identical interface against the future application. After the change each script takes four arguments:

1. The path to the directory in which the output will be saved
2. The path to the directory containing information about the ligand/s
3. The path to the directory containing information about the protein/s
4. The path to the design file

The other mentioned variables has now gotten static names, e.g. the path to the file containing the PDB codes for the complexes that will be docked, from now on this file always lies in the directory in which the output will be saved and has a specified name. After discussion with end users it was decided that future docking program should be possible to run using these set of arguments to the controlling scripts.

5.2.2 The remaining scripts

The remaining scripts were rewritten so they could take arguments, in order to facilitate the communication between the application and the scripts. Furthermore, to make it possible to develop the procedure it was decided that the application should be able to run two arbitrary additional scripts. For more information about the scripts and their arguments, please see Appendix A.

5.3 The design

Each docking script takes a path to a design file as input. The design file consists of a number of rows, each one corresponding to a run of the docking program. Each row contains the values assigned for the parameters during that particular run. The design files are constructed using a number of statistical designs, the designs used are full factorial design, fractional factorial design, and D-optimal design, all described in Eriksson et al. [28]. To

construct a full factorial design all parameter values are combined with all other parameter values. Thus, a full factorial design covers the whole parameter space, i.e. all possible combinations of parameters.

Fractional factorial design, on the other hand, is designed to cover the parameter space in as few runs as possible. These are done by dividing the full factorial design in fractions, and then do the runs specified by one of these [28]. When implementing a fractional factorial design of arbitrary parameters a full factorial design is done based upon a subset of the parameters and then the values in the remaining columns are calculated based upon certain rules [29]. When constructing a D-optimal design, the best subset of experiments from a candidate set (a matrix X) is selected, using the criterion that the selected design should maximize the determinant of the matrix X'X for a given regression model [28].

It was decided that the application should offer the user the possibility to manufacture full factorial and fractional factorial designs. The fractional factorial design was further limited to using parameters with one or two values only; the number of parameters with two values for one design was reduced to eleven. The fractional factorial design implemented in the application is based upon the rules described by Box et al. [29]. Since there is a possibility that the user wants to use other types of designs, the application also has the possibility of supplying a previously generated design. This feature also has the advantage that the user do not need to generate a design each time a docking program will be run, opening up for the possibility of using one design several times.

5.4 The parameters

To be able to construct the design file the application need to handle the parameters for docking program it is about to run. Unfortunately, the parameters to the programs vary a lot, and can be as many as 100 specific parameters. The application must have some information about the parameters available in the program it is about to execute, otherwise the task would be impossible. Based on the parameter list for FRED the specification for a parameter file was constructed. In this file all parameters for the program is specified by name, type of parameter, default value and (if present) the restrictions on the parameter values. This file is given as input to the application, which reads it and manages the parameters and the design file resulting from the settings. The specification for the parameter file can be found in Appendix B.

6. The implementation

6.1 The Graphical User Interface

A desirable feature of the application was that is should have a graphical user interface (GUI) and, furthermore, it must be easy to use. Since the user should have the possibility of either making a design file using the application or supply a previously generated design file, the GUI has to adjust to the chosen action. In figure 3 the resulting flow chart of the program is visualized. The program starts and the user chooses program settings. If the user supplies a pregenerated design file the GUI now makes it possible for the user to choose setting for the handling of the output otherwise the user first makes the design settings then chooses the handling of the output. When this is done the user approves of the chosen settings and the program executes the chosen scripts.

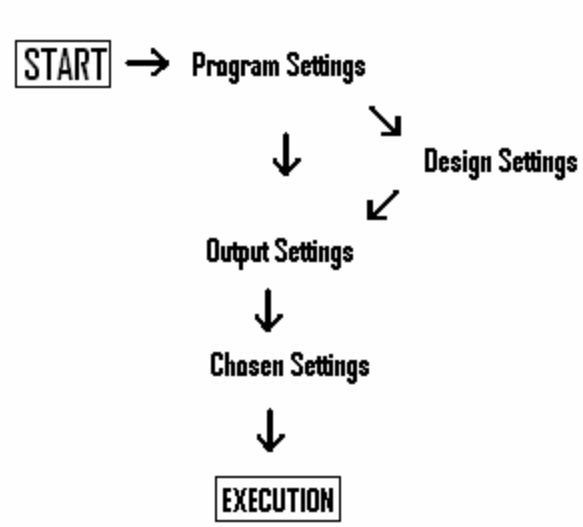


Figure 3. Flowchart describing the user input steps.

To make the graphical user interface as flexible as possible it was decided that the program should use a card layout showing different views corresponding to the program, design, output and chosen settings (figure 3). The layout had a further advantage: it also made it easier to ascertain that the user had supplied all necessary information for executing the program before it was possible to continue to the next view.

6.2 The visualization of the parameters

The task of visualizing the parameters to the user in an understandable way was complicated. The view did not only need to display all possible parameters (around 100) but also needed to make it possible to change the values of these parameters and display the chosen parameters and their values separately. In addition, all of the mentioned tasks needed to be fulfilled in a small area since the card layout takes the size of the biggest participating member. Since the number of experiments and other statistics for the design changes with the number of parameters in the design, the design statistics also needed to be displayed in the interface. The solution to this complex task was a view depicting information about the current design and a table containing information about the chosen parameters and their settings. When the user wants to choose parameters a dialog is opened depicting the parameters in a tree. The user can manage the tree by changing information in the parameter file described above and in Appendix B. When a user chooses a parameter another dialog is opened in which the user can make the settings for the chosen parameters. The interface and the dialogs are further explained and visualized in the design settings section.

7. The result – an application called DockControl

The finished application, called DockControl, communicates with the user using three subsequent dialogs. The first of the dialogs collects the parameter file, in the second all settings for the execution of the scripts are made and in the third the progress of the execution is shown. In the following section each of the dialogues will be depicted, the class relationships will be presented and a brief overview of each class will be made. In Appendix C a user manual can be found.

7.1 The class relationship

In figure 4 the different classes of the application is depicted. The classes are divided in

four main groups, A, B, C, and D, furthermore, there are five classes not belonging to any group. Group C is further divided into four subgroups, 1, 2, 3, and 4. The different groups controls specific parts of the program and will be further explained later in the text. The five classes that do not belong to any group are utility classes used by many of the other classes. The utility classes are the `FractionalDesignRules` class, the `SpringUtilities` class and three Exception classes. The Exception classes are used when an error occur while the application reads the parameter file. The `FractionalDesignRules` class is used whenever the application will construct a fractional factorial design. The `SpringUtilities` class, finally, are used when components are laid out in their containers. Group A contains the classes handling the program, the startup and the settings made by the user. Group B handles the dialog collecting the parameter file. Group C is the group handling the GUI in which all settings is made, each subgroup of the class collects different settings. Group D, finally, controls the execution of the docking program according to the settings.

7.2 The application

7.2.1 The start dialog

The Dialog displayed at the start of the program is shown in figure 5. In this dialog the user supplies the path to the parameter file and also the name of the docking program that will be used in this run. The class behind the dialog belongs to group B and is called `InputFrame`. Upon starting the DockControl program the `DockControl` class (group A) is instantialized, this class in turn instantializes and displays the dialog which is managed by the `InputFrame` class. When the needed information has been entered into the dialog and the user has pressed OK the InputFrame instance gives the DockControl instance the information whereupon the DockControl instance disposes of the InputFrame instance.

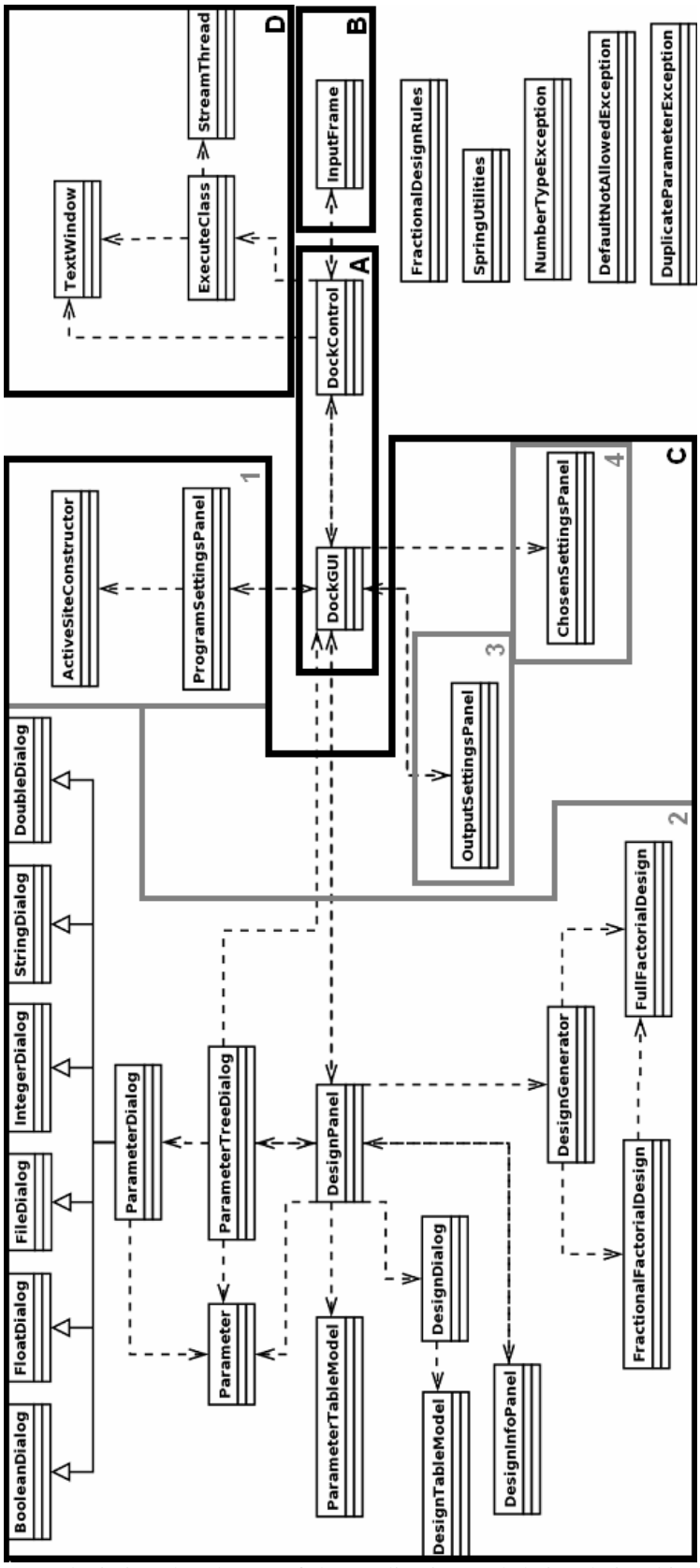


Figure 4. UML diagram depicting the classes of the application and their relationships. Group A contains the classes handling the program, the startup and the settings made by the user. Group B handles the dialog collecting the parameter file. Group C is the group handling the GUI in which all settings is made, each subgroup of the class collects different settings. Group D controls the execution of the docking program according to the settings.

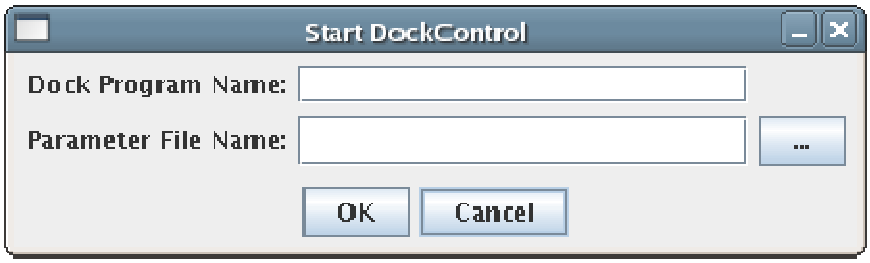


Figure 5. The dialog displayed upon starting DockControl.

7.2.2 The settings dialog

The information DockControl has gotten from the InputFrame instance is used when instantiating the `DockGUI` class (group A). The class sets up a GUI which has four different views, each of these corresponding to a subgroup of group C. DockGUI stores the settings made by the user in each view, and controls when it should be possible for the user to change view. The different views of the settings dialog is the program settings view (controlled by group C1), the design settings (group C2), the post treatment of data (group C3) and finally, the chosen settings (group C4).

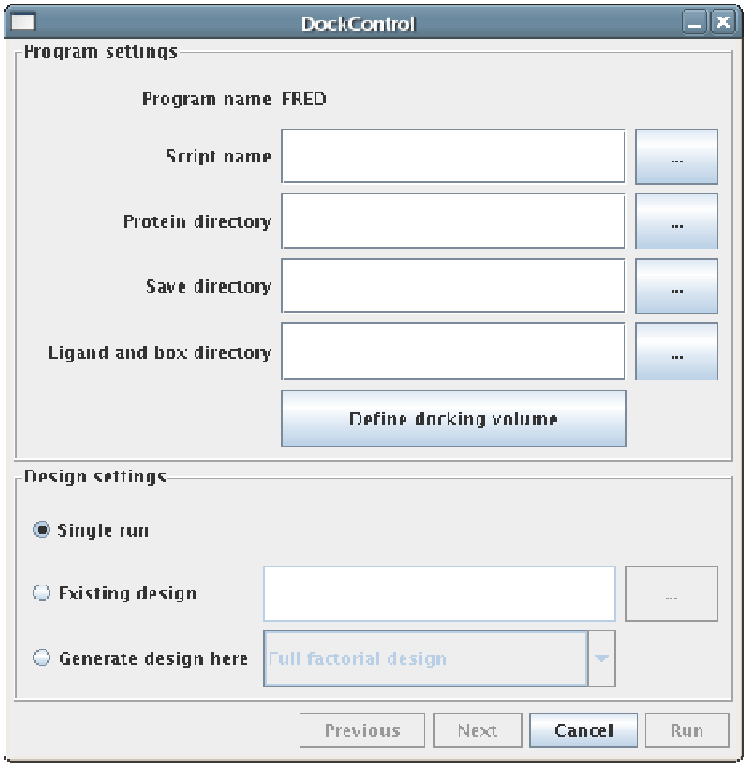


Figure 6. The first view of the DockControl GUI, in the text called the program settings view.

In figure 6 the first view of the settings dialog is shown. The `DockGUI` class controls the part of the GUI displaying the “Previous”, “Next”, “Cancel” and “Run” buttons. Each time a user has supplied all needed information for a specific view the class controlling this view notifies the `DockGUI` class, which, in turn, enables the “Next” button, thereby controlling

that all information needed for doing a run is supplied. The “Cancel” button is always enabled and so is the “Previous” button, displaying the previous view, with the exempt of when DockGUI instance is displaying the first view. The “Run” button is enabled only when the last view is displayed. Note that this view demands no settings being specified.

When the “Run” button is pressed the DockGUI instance sends the chosen information as two dimensional arrays to the DockControl instance.

The program settings view

In the first view of the application the user chooses input to the docking program, the view is depicted in figure 6. The uses supplies the docking program script, the directory containing information about the protein/s, the directory in which the output of the docking program will be saved and the directory containing information about the ligand/s.

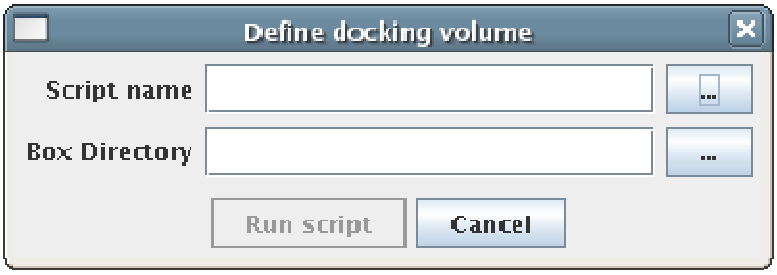


Figure 7. The active site dialog.

Underneath the square in which to supply the directory containing information about the ligand/s, there is a button called “Define docking volume”. The docking volume is a volume enclosing the area in the protein where the ligand will be docked. The file containing the area description can be constructed using a script, so when the “Define docking volume” button is pressed a dialog (displayed in figure 7) shows up. In this dialog the name of the script constructing the docking volume and the directory containing information about the ligand/s should be supplied. Once the “Run script” button is pressed the script is executed and the resulting file is stored in the given directory.

The user also defines which kind of parameter design the current docking session will use. Either the user can choose to do a single run (called “Single run”), use a previously constructed design (called “Existing design”) or use the application to construct a design (called “Generate design here”); in the latter alternative the user also needs to choose a full factorial or a fractional factorial design.

The classes managing the program settings view and their relationships belong to group C1. The `ProgramSettingsPanel` is the class managing the program settings panel and also saving all information the user supplies. Once the user has supplied all information to this view the ProgramSettingsPanel instance sends the information to the DockGUI instance. The `ActiveSiteConstructor` is the class managing the dialog that appears when the “Define docking volume” is pressed and executing the script constructing the wanted file. Once the user presses the “Run script” the DockGUI instance is notified and gets the name of the directory of the active site file from the ActiveSiteConstructor instance.

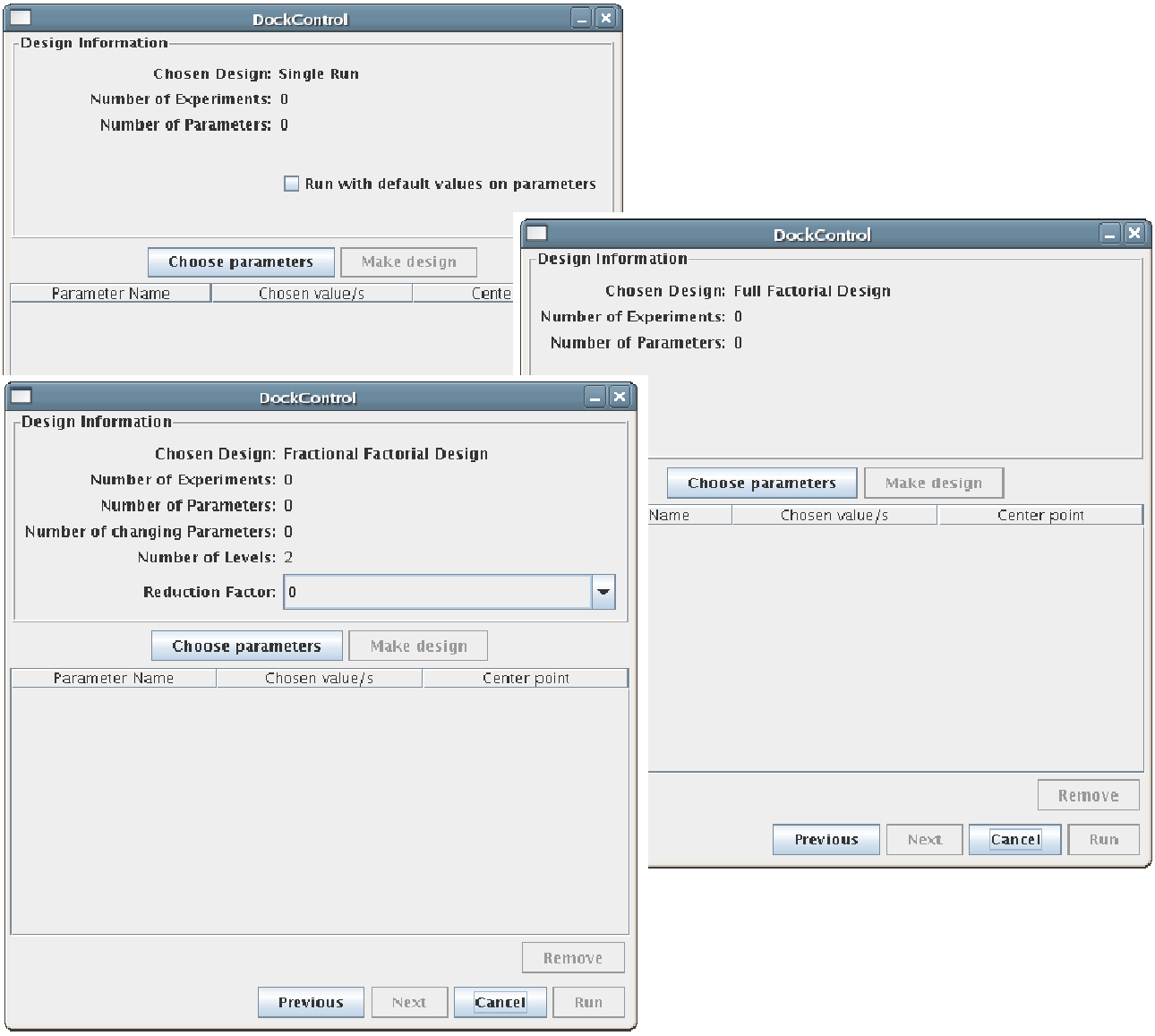


Figure 8. The three design views. From the top, single run, full factorial design and fractional factorial design.

The design settings

If the user has not supplied a previously constructed design, the application will be used to make one. The view used when the user chooses which parameters to include in the design is depicted in figure 8. The figure shows three different windows, corresponding to the three different design choices, single run, full factorial design or fractional factorial design, respectively. Each view is divided in two parts, the upper part informs the user of the statistics for the current design and the lower part displays the chosen docking program parameters.

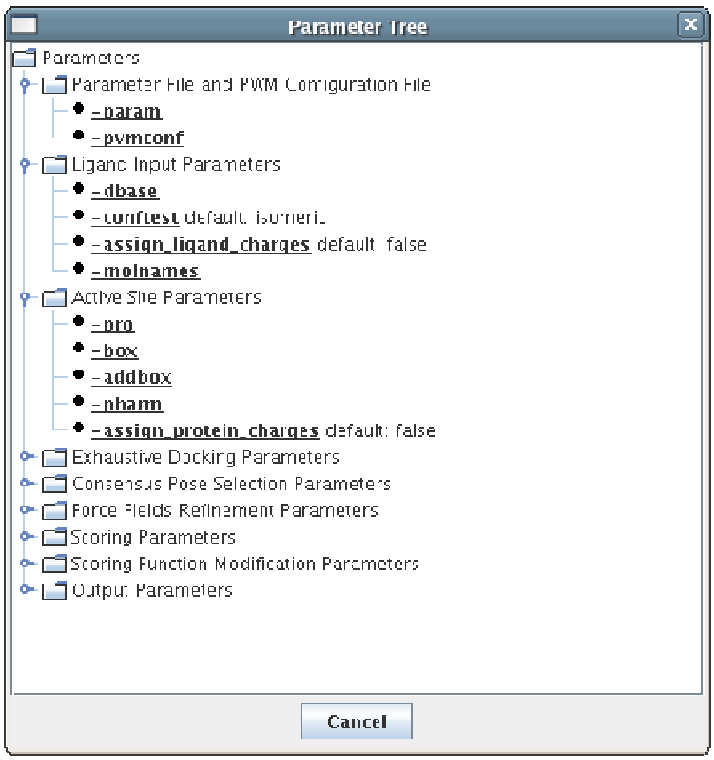


Figure 9. The dialog displaying the parameters in a tree according to the parameter file. The default value for the specific parameter is also shown, given that the parameter has one.

When a design will be made, the user first chooses the parameters that will be used in the design. The parameter choices are made by first pressing the “Choose parameters” button, whereupon a dialog will appear, displaying a tree containing the parameters present in the parameter file (figure 9). When a parameter is chosen a dialog appears in which the user chooses the value/s this parameter will have. In figure 10 some of the possible parameter dialogues are shown. The dialogues differ depending on what kind of design is chosen, single run only can take a single value for each parameter, in figure 10 the dialogues corresponding to a fractional factorial design is displayed.

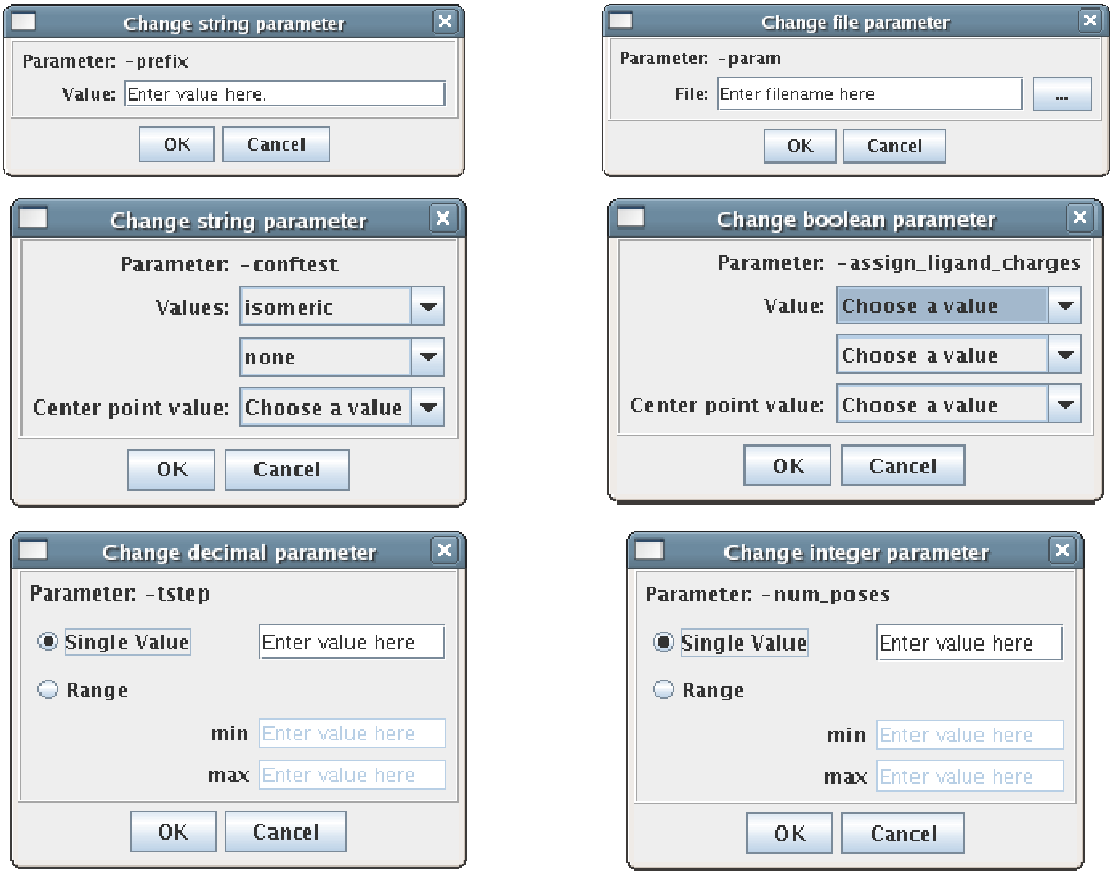


Figure 10. Parameter dialogues belonging to fractional factorial design. From the upper left corner to the right and downwards: *string* parameter without predefined values, *file* parameter, *string* parameter with predefined possible values, *Boolean* parameter, *float* parameter, and *integer* parameter. In addition to this a dialog for *double* parameters exists but it is not shown.

When the parameters of interest are chosen the “Make design” button is enabled and when pressed a frame showing the completed design in a table will appear (figure 11). The user now has the possibility of saving or discarding the design. To be able to continue to the next view the user has to save one design.

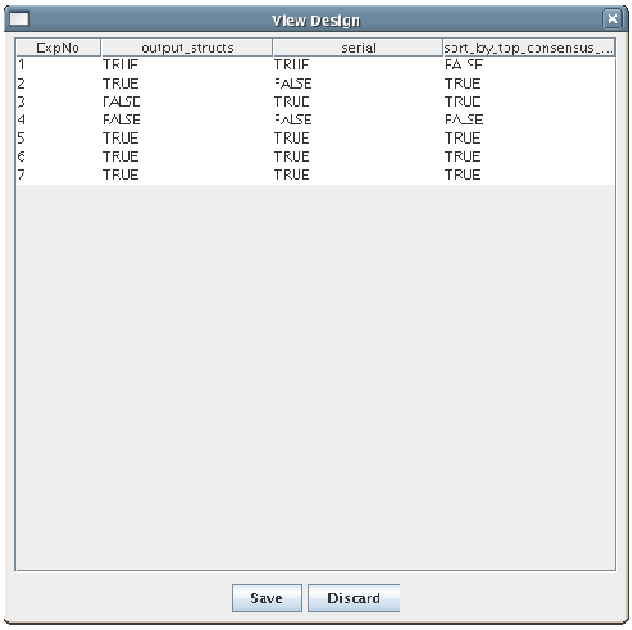


Figure 11. An example on a design constructed by the application. This is a fractional factorial design.

The design construction and the parameter handling is the most complex part of the application; consequently, it is handled by the biggest subgroup of group C, group C2. The `DesignPanel` class handles the basic view together with the `DesignInfoPanel` class. The latter handles the design statistics shown in the upper part of the design view. The table displaying the chosen parameters and their values is managed by the `ParameterTableModel` class.

The dialog displaying the parameters of the current docking program is handled by the `ParameterTreeDialog` and the individual parameter dialogues are handled by the `ParameterDialog` class and its children. The information about each parameter is stored in a Parameter object. The `DesignGenerator` class is constructing the design using the `FractionalFactorialDesign` and `FullFactorialDesign` classes. In the case of a fractional factorial design, the `FractionalDesignRules` class is also used.

Finally, the dialog displaying the finished design is handled by the `DesignDialog` class, with the `DesignTableModel` handling the table in which the values of each parameter is shown.

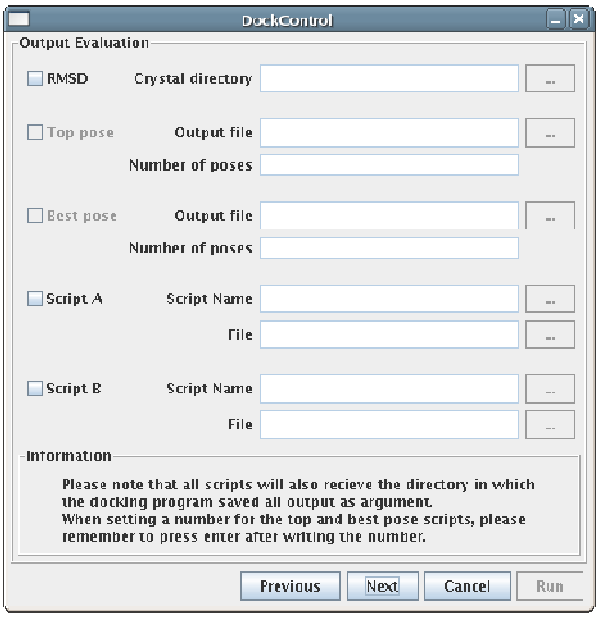


Figure 12. The panel in which the treatment of the data resulting from the docking run/s can be chosen.

The post treatment of data

In this step the treatment of the data resulting from the docking run/s of the docking program can be chosen. As can be seen in figure 12, the possible choices are to calculate RMSD, to extract top and/or best pose, and there is also the possibility to use up to two other scripts. The panel is handled by one single class, the `OutputSettingsPanel` class (group C3).

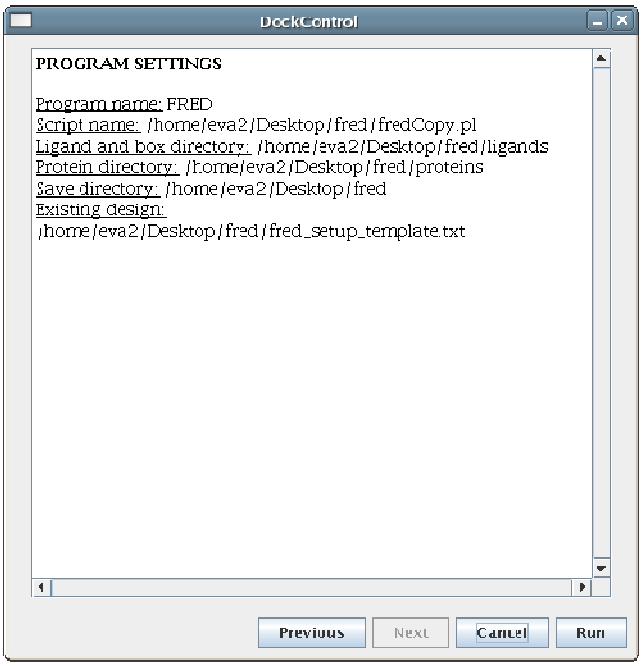


Figure 13. The settings chosen by the user displayed.

The chosen settings

Finally, the settings chosen by the user is displayed. In figure 13, the user only has chosen to execute the program without handling the output in any way after the run. This feature is also handled by only one class, the `ChosenSettingsPanel` class (group C4).

7.2.3 The Execution of the program

The classes handling the execution of the docking programs and the different scripts belong to group D. Once the user has pressed "Run" in the final view of the DockGUI the execution of scripts starts. The scripts are executed in the following order: docking program script, RMSD script, top pose script, best pose script, and, finally, the two arbitrary scripts mentioned in section 3.2.2, given that all are chosen by the user. As can be seen two classes controls the execution, one of them, `CommandExecuter`, runs the script, the other, `ProgressViewer`, displays the output of the scripts and also informs the user of the progress. An example of how `ProgressViewer` can display the progress is shown in figure 14.

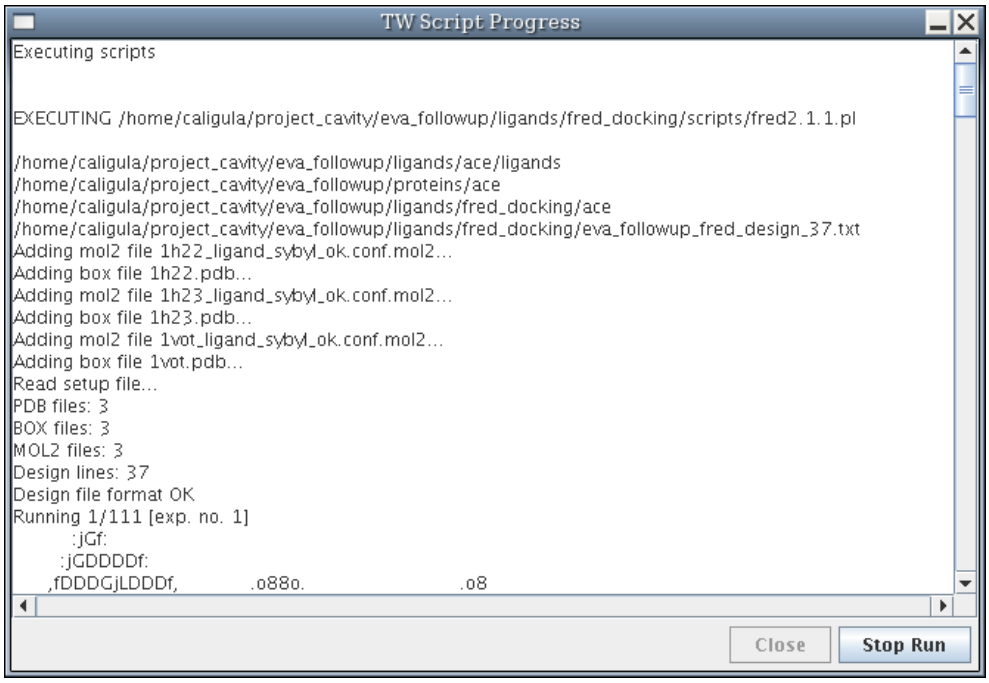


Figure 14. The display of the script execution progress.

8 DockControl applied to FRED

As a part of the thesis a number of parameters were analyzed using the project methodology previously described, using the DockControl software to construct the design file and controlling the docking run. The docking program chosen for this experimental run was FRED, six parameters were chosen to be analyzed, and all of these affect the docking step in some way. The result of the run was analyzed using SIMCA-P+ software [25]. The complex used for this run was HIV protease co-crystallized with 30 different ligands. HIV protease enzyme is a protein involved in the life-cycle of the HIV virus; the protease cleaves polypeptides, and affects the ability of HIV virus to infect cells. Hence, the protease is an attractive target for anti-HIV drugs.

8.1 Preparation of the complexes

30 ligand-protein HIV protease complexes were chosen from the PDBbind database [30, 31] and was prepared for docking according to the procedure described by Lindström et al. [32]. In Appendix D the different ligands and their conformations can be seen.

8.1.1 Settings for the run

The docking was made using FRED software and a fractional factorial design of docking parameters with one center point, constructed using DockControl software. The latter was also used to run the docking. The design is visualized in table 2. The parameters modified during the run of this design were exhaustive_scoring, tstep, rstep, clash_scale and neg_img_size. exhaustive_scoring controls which scoring function is used during a specific experiment, in this design the scoring function is varied between chemgauss and shapegauss scoring functions. The two parameters rstep and tstep control the rotational and translational stepsize of the exhaustive search. The clash_scale parameter defines when the protein and ligand are considered to clash and, finally, the neg_img_size controls how distant atoms are allowed to be places from the protein.

ExpNo	exhaustive_scoring	tstep	rstep	clash_scale	neg_img_size
1	shapegauss	1.0	1.0	0.75	xlarge
2	shapegauss	1.0	1.5	0.75	normal
3	shapegauss	1.5	1.0	0.25	xlarge
4	shapegauss	1.5	1.5	0.25	normal
5	chemgauss	1.0	1.0	0.25	normal
6	chemgauss	1.0	1.5	0.25	xlarge
7	chemgauss	1.5	1.0	0.75	normal
8	chemgauss	1.5	1.5	0.75	xlarge
9	shapegauss	1.25	1.25	0.5	large

Table 2. The parameter design for the run.

8.2 Data treatment

After the run the RMSD for each experiment and complex were calculated, and the best pose and top pose data for each experiment was extracted. A median and average RMSD of best and top poses, respectively, was calculated (table 3). The resulting data was analyzed using SIMCA-P+ software [25].

ExpNo	Best pose median	Top pose median
1	4.74	7.485
2	4.02	7.155
3	4.405	7.095
4	3.885	6.735
5	5.355	7.185
6	4.51	7.415
7	5.07	7.16
8	4.055	7.265
9	4.375	7.415

Table 3. The median for best and top pose for each experiment.

8.3 Analysis

In SIMCA-P+ PLS regression was used to construct a model describing the dependencies between the calculated RMSD median values and the parameter settings. The qualitative parameters (i.e. echaustive_scoring and neg_img_size) were transformed into Boolean parameters, if they were used in the experiment they were assigned 1 if they not were used they were assigned value 0. The model constructed described the relationship between the parameters and both the best and top pose median. Linear-, cross- and square terms of the parameters were used to construct the models. Non-significant terms were excluded iteratively to formulate the final model. No linear terms were excluded, even if non-significant.

Score plots, loading plots, R2 values, Q2 values, observed versus calculated values plots, residuals plots, and validation plots were used to interpret and validate the model. The score plot (figure 15) is a map of the observations, showing the possible presence of outliers, groups and patterns in the data. The loading plot (figure 16) is a plot describing the major relations between the variables (here parameters) and the responses (here median rmsd values). The R2 value explains how much of the variation in the variable matrix that can be explained by the model, the Q2 value describes how well a variable within the model can be predicted. The observed vs. calculated values plot (figure 17) shows exactly what it is called. For a good model all the points should fall close to the 45 degree line. The residuals (un-modeled data) of the model are plotted in a residual plot, found in figure 18. The validation plot (figure 19) assess the risk that the model is spurious, that is that the model just fits the training set well but does not predict the median for new experiments. Here the R2 and Q2 values of the original model is compared with the goodness of fit of several models based on data where the order of the responses (i.e. the order of the numbers in the response matrix) has been randomly permuted, while the parameter matrix has been kept intact.

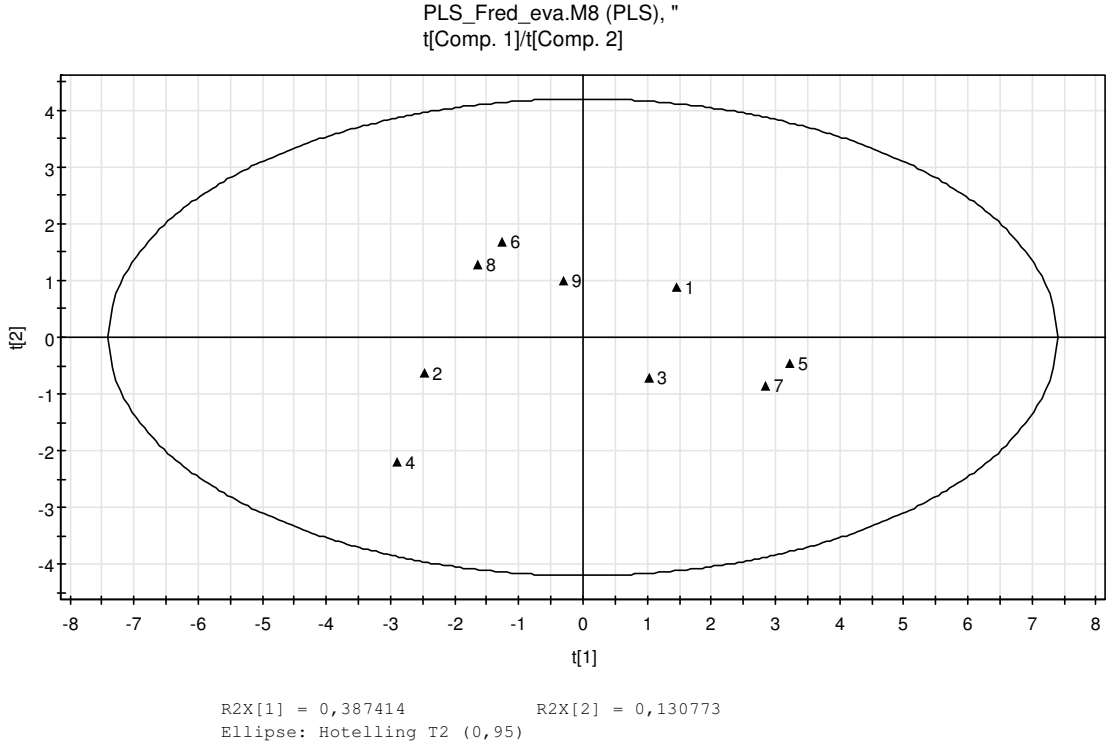


Figure 15. Score plot for top and best pose model.

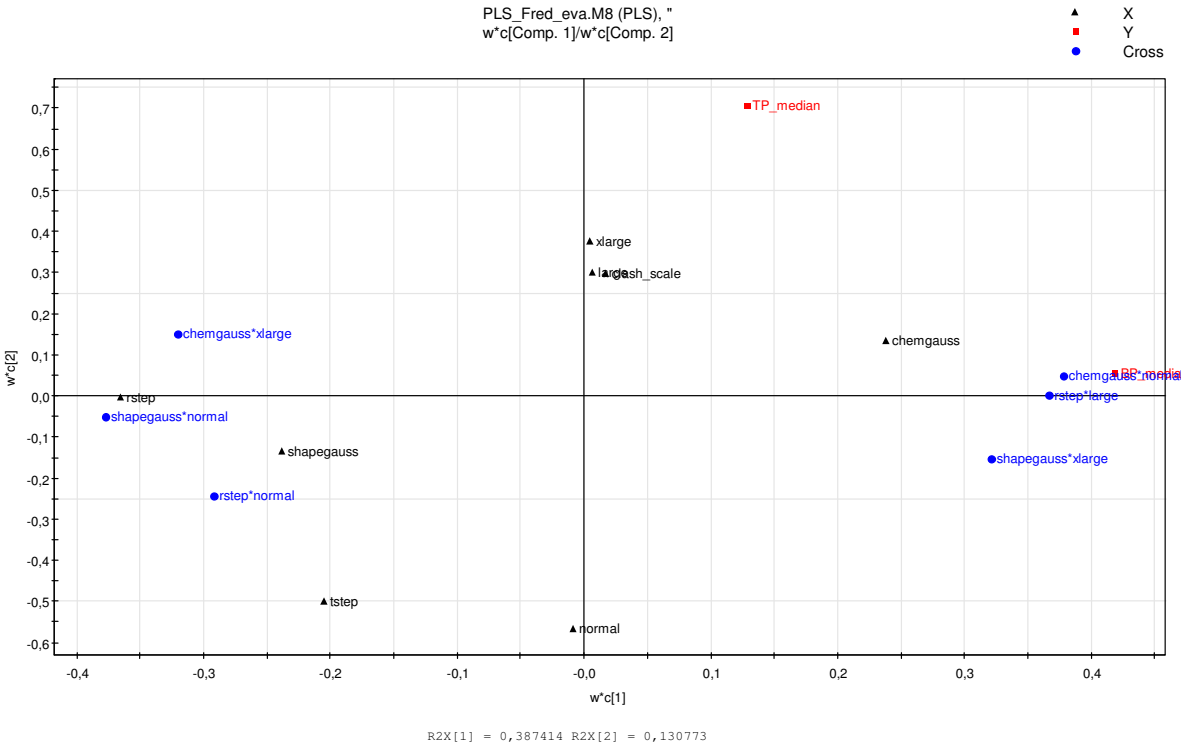


Figure 16. Loading plot for the top and best pose model.

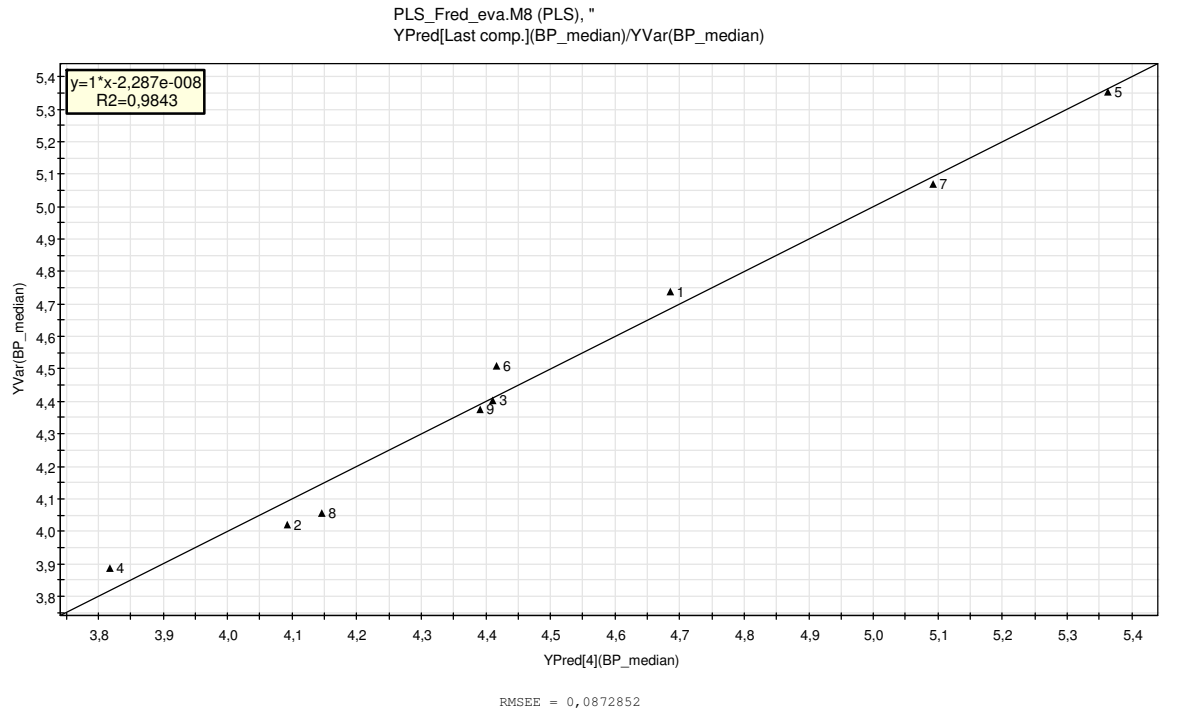


Figure 17. Observed vs. calculated values for the top and best pose model.

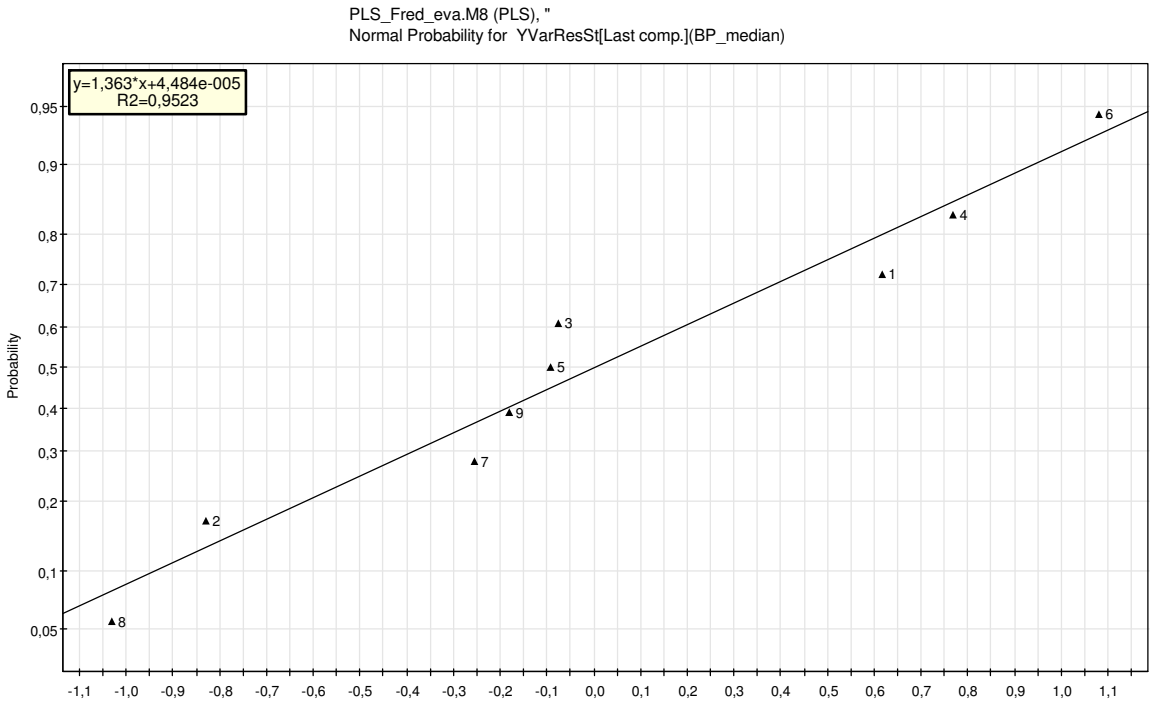


Figure 18. A plot of the residuals for the top and best pose model.

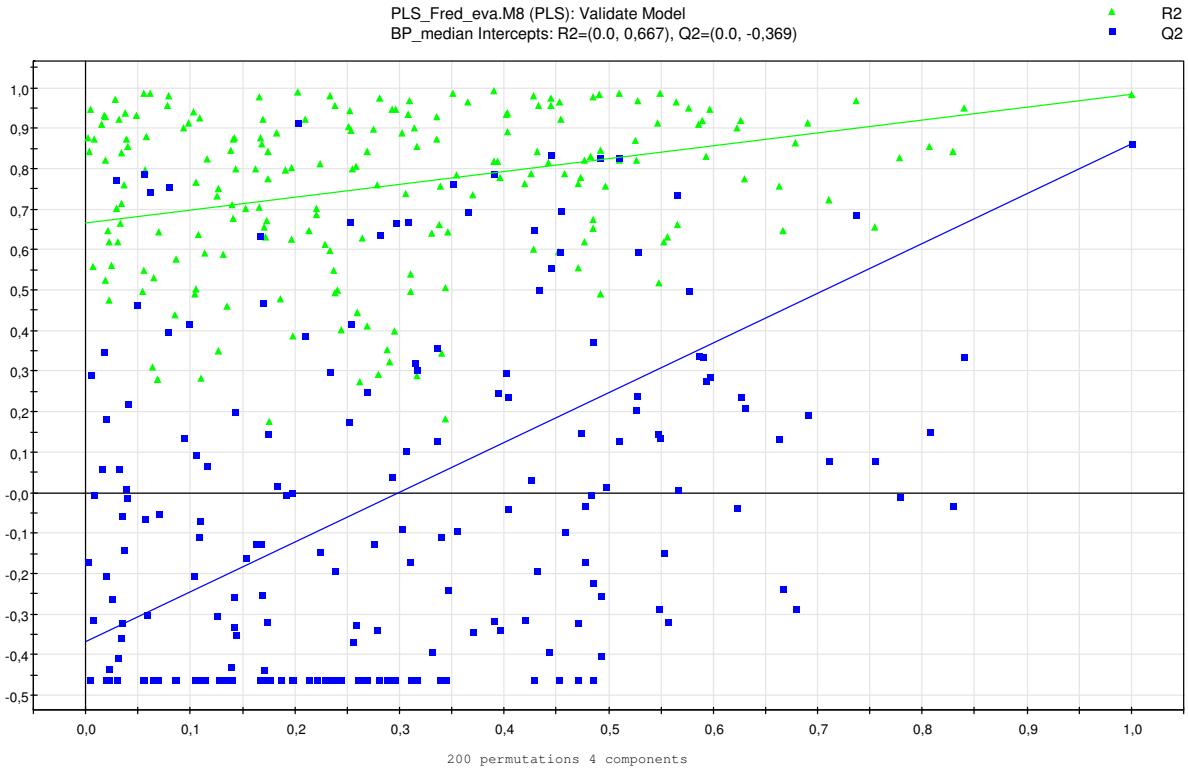


Figure 19. The validation plot (with 200 permutations) for the top and best pose model.

8.3.1 Best and top pose model

The experiment with the lowest median for both top and best pose is experiment 4 (table 3), the other experiments are ranked, with the best one first, according to 2, 8, 6, 9, 3, 1, 7, and

5. The score plot (figure 15) shows no groupings of data, nor is any outliers visible. An experiment that will have a good result (figure 16) will have a high value for rstep and a slightly lower for tstep. The best scoring function is shapegauss and the neg_img_size should be normal, even though the difference between normal and the other values, xlarge and large, is miniscule. The final parameter, clash_scale should be given a low value.

The model can explain the variation in the variables, the R2 value is 0.982, and has an acceptable internal predictable capability, with a Q2 value of 0.752. The plot of observed vs. calculated values (figure 17) and the plot of the residuals (figure 18) shows that the values have an approximately but not perfect linear relationship. This gives a hint that the model is not ideal, i.e. the relationship between the variables and the rmsd outcome is a linear relationship. A validation experiment can be seen in figure 19. If the same set of experiments are used and the calculated medians are moved between the experiments in a random fashion it is possible to achieve better prediction of variables and better explanation of variable variation using the same model. Thru the validation experiment a weakness in the model is revealed; the small apan in the Y matrix (table 3), which makes its predictive capability weak.

8.3.2 Comparison to other models

In addition to the above model two further models have been constructed, one for explaining the relationship between the top pose medians and one for explaining the best pose medians (data not showed). The additional models did not differ in any profound way from the already described model. One way in which they did differ was what the preferred values for the neg_img_size and clash_scale: in order to get low RMSD on the top pose the neg_img_size value should be normal, while a xlarge value correspond to a low RMSD for the best pose, in the clash_scale case, furthermore, a low RMSD for the top pose correspond to a low value while a good RMSD for the best pose is achieved by a high value. None of the three models state clearly that any of the parameters should have a high or low value, instead the parameters lies close to the line separating the parameters with preferred high values from the parameters with preferred low values. Neither the top pose model nor the best pose model did find a clear relationship between the value of the parameters and the calculated medians.

8.4 Discussion of the analysis of the constructed models

When evaluating the models resulting from this experimental design it must be remembered that the resulting median differs very little between the single experiments. This small span in the response values used in the PLS modelling makes it hard to find valuable relations and correlations between the parameters and the resulting output. If any more runs were to be made based on analysis made here, the new design should use the same variables and values but use more experiments to get a better spread on the resulting medians. In particular, more experiments should be performed in order to evaluate the preferred clash_scale and neg_img_size values.

9 Discussion

The aim of this thesis was to produce a program which could control a number of scripts without the need to modifying the scripts or move between computers for running them. The application should control differing docking programs, and provide the user with an easy-to-understand interface, through which the needed docking, statistical experimental design and the post-docking analysis could be made. These specifications are fulfilled. The software produced, called DockControl software, can run scripts controlling the two docking programs FRED and GOLD now available to the research group. The software presents the opportunity for the user to make either one docking run with specific parameter settings or runs based on

full factorial or fractional factorial designs. The user can also choose between no post-docking handling of the data or RMSD calculations and extraction of best and top poses. Furthermore, it is possible for the user to modify the project methodology now used and in the future use other scripts in the methodology. The DockControl software fulfils all the basic requirements mentioned.

In section 2.3 a number of further requirements were made. These were that the application should make it possible to save time, be easy to use, be robust, be flexible, be well documented and be platform-independent. Unfortunately, it has not been possible to make a comparison between the time consumption for the realization of the methodology as it looked before and the time consumption when using the DockControl software. Intuitively, however, the application should save time for the user, which is also the belief of the researchers for which the application is developed. Using DockControl, the user no longer need to manually modify a range of script before every docking run, neither do the user has to sort and move docking output data between different folders and computers. The application had been written to be as robust as possible, and thorough testing has been made. Every input of the user is thoroughly tested before it is used by the application. As long as the correct information is written in the file describing the parameters, and the executed scripts are stable, no problems should arise. Since the application is written in Java it is possible to move it between different platforms: all testing of the application has however been made in Unix/Linux environment and even though the application uses no platform specific commands, unforeseen problems could arise when running the application in another environment. The application has been well documented and future developers wanting to modify the application should have no problem understanding or change the code.

The application is built to be flexible, but as the flexibility is based upon two docking program there is a possibility that it will encounter problems when trying to run upgraded versions of FRED or GOLD or other docking programs. Another requirement concerning flexibility of the application was that it should be possible to only use parts of the program. In figure 20 the different usages of the application is shown. The application can be used to make a file describing the position of a protein active site, run a docking program with or without analyzing the output and make a statistical experimental design. Note that these tasks can with ease be performed using the original Perl scripts as well, for the statistical experimental design there are more specialized software with more choices of design characteristics. The application is best when all parts are used, not when only some features will be exploited.

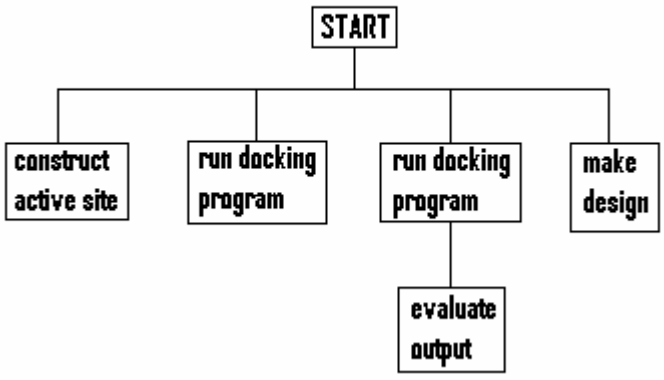


Figure 20. The application can be used to construct a file describing the active site, to run a docking program , to run the docking program and to evaluate the output of it and to construct a design file.

During the development phase the GUI of the application has been discussed with the supervisor to ensure that the labeling and the design of the GUI are intuitive. It is, however, hard to say if the application is user-friendly until a number of users has been using the application and made a number of dockings. All efforts have been made to make the GUI as user-friendly and intuitive as possible.

The application has a feature that needs to be implemented in the scripts; the user will be able to stop the execution of the scripts by pressing a button in the GUI. For the stop function to work the scripts need to maintain information about all processes they start to be able to kill these if the user chooses to stop the execution. This is perhaps not ideal since errors can be introduced by the script developer. On the other hand, this was the only possible solution for the user to have the possibility of stopping the execution.

The application needs one document, aside from the scripts, to function. It is the parameter file, a document describing the parameters for the docking program the user wants to run during the current docking session. The parameter file is a quite simple file but it is the file which makes the difference between function and failure and it is the weak spot for the application when it comes to malfunctioning. It is also a document fully dependent on the user. It could be discussed if the application really is robust when relying on this document. On the other hand, the parameter file is not something that needs to be constructed for each new user, this file needs to be written once for every program which the application will control, modifications of the file needs to be done each time a new version of a docking program will be taken in use, in this case modifications is only needed if the parameter list has been modified in any way. The construction and modification of the parameter file should preferably be the responsibility of one user, in this case the every day user needs not to be bothered and the possibilities for user induced errors will be kept to a minimum.

10 The future

There are a number of measures that can be taken in order to improve DockControl. The types of parameters to use could be extended; it would be nice to have a dialog handling directory parameters or perhaps parameters taking longer strings. At the moment it does not seem to be needed but future docking programs might need these kinds of parameters. It would also be nice to have the possibility to make a D-optimal design or a fractional factorial design containing more than 11 parameters with two values each. Furthermore, it would be good for the structure of the program to have a common parent class for the `IntegerDialog`, `FloatDialog` and `DoubleDialog` classes. This should be a child class of the `ParameterDialog` class and hopefully, much common code between the three number classes would be saved. It is also possible to further extend the Parameter class so that it would get a child class for each type of parameter, but it is not certain that the application or the understanding of it would benefit from this solution.

11 Acknowledgements

I would like to thank a number of persons for their help during the work of my thesis.

Anna Linusson and David Andersson, my two supervisors at the Department of Chemisty, for always being there when I had a question. Anna, thank you for making it possible for me to do this thesis. David, without your help this thesis would not have been finished.

Jan-Erik Moström at the Department of Computing Science, who helped me when I encountered difficulties that I could not solve on my own.

Per Lindström, my supervisor at the Department of Computing Science.

Anders Vestlin, Malin Olsson and Johanna Bergh, for always believing in me and

supporting me when things get tough.

12 References

1 Patrick, G. From concept to market. In *Instant Notes in Medicinal Chemistry;* Bosher, A. Eds.; BIOS Scientific Publishers Limited: Oxford, UK, 2001; pp 4-6.

2 Leach, A. R.; Shoichet, B. K.; Peishoff, C. E. Prediction of Protein-Ligand Interactions. Docking and Scoring: Successes and Gaps. *J. Med. Chem.* **2006,** *49(20),* 5851-5855.

3 Kitchen, D. B.; Decornez, H.; Furr, J. R.; Bajorath, J. Docking and Scoring in Virtual Screening for Drug Discovery: Methods and Applications. *Nat. Rev. Drug Discovery.* **2004,** *3,* 935-949.

4 Jones, G.; Willett, P.; Glen, R. C. Molecular Recognition of Receptor Sites using a Genetic Algorithm with a Description of Desolvation. *J. Mol. Biol.* **1995,** *245,* 43-53.

5 Jones, G.; Willett, P.; Glen, R. C.; Leach, A. R.; Taylor, R. Development and Validation of a Genetic Algorithm for Flexible Docking. *J. Mol. Biol.* **1997**, *267,* 727-748.

6 *FRED - Fast Rigid Exhaustive Docking*, version 2.1.2; OpenEye Scientific Software: Santa Fe, NM, 2004.

7 *MOE - Molecular Operating Environment*, version 2006.08; Chemical Computing Group Inc.: Montreal, Canada, 2004.

8 Morris, G. M.; Goodsell, D. S.; Halliday, R. S.; Huey, R.; Hart, W.E.; Belew, R. K.; Olson, A. J. Automated Docking Using a Lamarckian Genetic Algorithm and an Empirical Binding Free Energy Function. *J. Comput. Chem.* **1998,** *19(14),* 1639-1662.

9 Sousa, S. F.; Fernandez, P.A.; Ramos, M. J. Protein-Ligand Docking: Current Status and Future Challenges. *Proteins: Struct., Funct., Bioinf.* **2006,** *65,* 15-26.

10 Ferguson, D. M.; Raber, D. J. A New Approach to Probing Conformational Space with Molecular Mechanics: Random Incremental Pulse Search. *J. Am. Chem. Soc.* **1989,** *111,* 4371-4378.

11 *OMEGA*, version 2.1.0; OpenEye Scientific Software: Santa Fe, NM, 2004.

12 Solis, F. J.; Wets, R. J.-B. Minimization by Random Search Techniques. *Math. Oper. Res.* **1981,** *6(1),* 19-30.

13 Tøndel, K.; Anderssen, E.; Drabløs, F. Protein Alpha Shape (PAS) Dock: A new gaussian-based score function suitable for docking in homology modelled protein structures. *J. Comput. Aided Mol. Des.* **2006,** *20,* 131-144.

14 Bursulaya, B. D.; Totrov, M.; Abagyan, R.; Brooks III, C. L. Comparative study of several algorithms for flexible ligand docking. *J. Comput. Aided Mol. Des.* **2003**, *17,* 755-763

15 Kellenberger, E.; Rodrigo, J.; Muller, P.; Rognan, D. Comparative Evaluation of Eight Docking Tools for Docking and Virtual Screening Accuracy. *Proteins: Struct., Funct., Bioinf.* **2004,** *57,* 225-242.

16 Warren, G. L.; Andrews, C. W.; Capelli, A.-M.; Clarke, B.; LaLonde, J.; Lambert, M. H.; Lindvall, M.; Nevins, N.; Semus, S. F.; Senger, S.; Tedesco, G.; Wall, I. D.; Woolven, J. M.; Peishoff, C. E.; Head, M. S. A Critical Assessment of Docking Programs and Scoring Functions. *J. Med. Chem.* **2006,** *49,* 5912-5931.

17 Andersson, D.; Thysell, E.; Lindström, A.; Bylesjö, M.; Raubaucher, F.; Linusson, A. A Multivariate Approach to Investigate Docking Parameters' Effect on Docking Performance. *J.Chem. Inf. Model* **2007**, *47(7)*, 1673-1687.

18 Leach, A. R. *Molecular modelling: principles and applications,* 2nd ed.; Pearson Education Limited: Harlow, England, 2001.

19 Lipinski, C. A.; Lombardo, F.; Dominy, B. W.; Feeney, P. J. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Adv. Drug. Deliv. Rev.* **1997,** *23,* 3-25.
20 Leo, A.; Hansch, C.; Elkins, D. Partition coefficients and their uses. *Chem. Rev.* **1971,** *71,* 525-616.
21 Moriguchi, I.; Hirono, S.; Liu, Q.; Nakagome, I.; Matsushita, Y. Simple Method of Calculating Octanol/Water Partition Coefficient. *Chem. Pharm. Bull.* **1992,** *40,* 127-130.
22 Moriguchi, I.; Hirono, S.; Nakagome, I.; Hirano, H. Comparison of Reliability of log *P* Values for Drugs Calculated by Several Methods. *Chem. Pharm. Bull.* **1994,** *42,* 976-978.
23 Nayal, M.; Honig, B. On the Nature of Cavities in Protein Surfaces: Application to the Identification of Drug-Binding Sites. *Proteins: Struct., Funct., Bioinf.* **2006,** *63,* 892-906.
24 Gohkle, H.; Klebe, G. Approaches to the Description and Prediiction of the Binding Affinity of Small-Molecule Ligands to Macromolecular Receptors. *Angew. Chem. Int. Ed.* **2002,** *41,* 2644-2676.
25 *SIMCA-P+*, version 11.0; Umetrics AB: Umeå, Sweden, 2005.
26 Fedora Eclipse. http://sourceware.org/eclipse/ (accessed Aug 1, 2007)
27 Wigglesworth, J.; McMillan, P. *Java™ Programming: Advanced Topics,* 3rd ed.; Muroff, J., Jones, S., Poirer, A., Eds.; Course Technology: Boston, Massachusetts, 2004.
28 Eriksson, L.; Johansson, E.; Kettaneh-Wold, N.; Wikström, C.; Wold, S. *Design of Experiments - Principles and Applications.* Umetrics AB: Umeå, Sweden, 2000.
29 Box, G. E. P.; Hunter, W. G.; Hunter, J. S. Table 12.15. In *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building.* John Wiley & Sons, Inc.: New York, 1978; p 410.
30 Wang, R.; Fang, X.; Lu, Y.; Wang, S. The PDBbind Database: Collection of Binding Affinities for Protein-Ligand Complexes with Known Three-Dimensional Structures. *J. Med. Chem.* **2004,** *47(12),* 2977-2980.
31 Wang, R.; Fang, X.; Lu, Y.; Yang, C.-Y.; Wang, S. The PDBbind Database: Methodologies and Updates. *J. Med. Chem.* **2005,** *48(12),* 4111-4119.
32 Lindström, A.; Pettersson, F.; Almqvist, F.; Berglund, A.; Kihlberg, J.; Linusson, A. Hierarchical PLS Modeling for Predicting the Binding of a Comprehensive Set of Structurally Diverse Protein-Ligand Complexes. *J. Chem. Inf. Model.* **2006,** *46,* 1154-1167.

Appendix A: Information about the scripts used in DockControl

The function of DockControl is based upon the execution of scripts. During each docking a run a total of seven scripts can be executed. In this appendix the requirements on these scripts, from the applications point of view, is stated. The seven scripts mentioned are: the script executing the docking program, the script constructing the files describing the docking volume, the RMSD script, the top pose script, the best pose script and two scripts here denoted A and B. The A and B script has no specific function but the possible execution of these are added to make the application more flexible to future changes of the project methodology.

The application serves each of these scripts with predefined arguments, as a consequence future new scripts executing a docking program, calculating RMSD etc. must be able to take these arguments.

A-1 General requirements on the scripts

The application can execute perl and python scripts. The scripts are executed using "perl scriptName arguments" or "python scriptName arguments". The output from each script (except the docking volume script) is shown in a window. The output from both STDERR and STDOUT is read and exposed to the user. If the script writes both to STDERR and STDOUT the output could be visualized in the wrong order, therefore it is recommended to write to one output, preferably STDERR, since this stream is unbuffered. It is worth noticing that if the script writes both to STDERR and STDOUT the user will not be able to discern which stream is currently written by reading the output on the text window supplied by the application.

A number of arguments to the script consist of names of directories. The directory name supplied from java does not end with "/" however, since it is required by the scripts that a directory name ands with "/", this character must be appended to the arguments consisting of directory names.

Please observe that the scripts will be called in the order:

2. programscript
3. rmsdscrpt
4. topscript
5. bestscript
6. supplementary scripts (in the GUI called script A and script B, script A is executed before script B)

Today there exist no possibility for the user to affect the order of execution. The script constructing the docking volume is executed when the user presses the button for execution, hence it is executed before and independent of the other scripts.

A-1.1 The script controlling the docking program

The application gives the script controlling the docking program four arguments, the name of the directory containing the ligand and docking volume files, the name of the directory containing the protein files, the name of the project directory, and the name of the design file. The arguments are given in the order they are mentioned. When the user wants to make a run with the docking parameters set to default values the name of the design file will be an empty string, the script will need to be able to handle this.

A-1.2 The script defining the docking volume

The script defining the docking volume is given two arguments, the name of the directory containing the crystal ligand files, and the name of the directory in which the docking volume files will be stored, in that order. No output from this script is shown to the user.

A-1.3 The RMSD, top pose and best pose scripts

These scripts differ from the other scripts in that the user can not supply the name of these scripts to the application; hence these three scripts have a predefined location and predefined names. The scripts must be found in a directory called “scripts” in the same directory as the application .jar-file is located. The scripts must be called rmsdscript, topscript and bestscript, respectively. The scripts can either be perl or python scripts, but if both a perl and python version of the scripts exists in the scripts directory, the application will execute the perl version.

The arguments given to the RMSD script is the name the directory containing the crystal files, and the name of the project directory, in that order. The top pose and best pose scripts get the name of the file to which the output form the current script will be written, the number of poses that the script will present for each complex and run, and the name of the project directory, also in that order.

A-1.4 Supplementary scripts

The application presents the user with the option of executing two supplementary scripts (in the GUI called script A and script B). The argument presented to these scripts is the name of a file and the name of the project directory, in that order.

A-2 General advice

When writing a script for the DockControl program, it is for the best to make sure the script function properly before trying to execute it using DockControl. If the script functions well outside of DockControl but not function when executed by DockControl, the best approach when debugging is find out the input to the script from DockControl, then executing the script by itself using said input. There is a possibility to get more informative error messages from the script when executing it without using DockControl than when it is executed by DockControl, and accordingly it is easier to fix the bug.

Appendix B: Specifications for the parameter file

The parameter file is the document describing the parameters for a given docking program, it is used by DockControl for the accurate construction of the design file and hence it is important that this file is accurate. The parameter file consists of two types of lines used by the application, the first type is the line describing a parameter, this line begins with a "-", the second type is the header lines, which starts with a number between 1 and 4. Line beginning with other sign than "-" or a number between 1 and 4 is ignored by the application, these lines can therefore be used for commenting or clarifying statements meant for the user.

A header line looks like:

```
[1,2,3,4] HeaderName
```

That is, a number (1,2,3 or 4) followed by the name of the header. The number denotes the level of the header, all no. 2 headers are sub headers of a no.1 header etc. An example:

The parameter file contains the following rows (for simplicity the parameter names are excluded):

1 First Header
2 Second Header
3 Third Header
4 Fourth Header
1 Fifth Header
1 Sixth Header
2 Seventh Header

The resulting tree will look as:

First Header
    Second Header
        Third Header
            Fourth Header
Fifth Header
Sixth Header
    Seventh Header

The header lines is not needed but could preferably be used to simplify the overview of the different parameters. A parameter belongs to the header directly above it in the parameter file; the concept is most easily described using an example:

If the parameter file looks like this:

1 Header 1
-parameter 1
-parameter 2
2 Header 2
3 Header 3
-parameter 3
1 Header 4

-parameter 4
-parameter 5
1 Header 5
-parameter 6

It gives a tree looking like this

Header 1
-parameter 1
-parameter 2
Header 2
Header 3
-parameter 3
Header 4
-parameter 4
-parameter 5
Header 5
-parameter 6

Each header corresponds to a node which can be expanded to display all subheaders and parameters of the expanded header. All level 1 headers is displayed at the beginning, to view level 2 headers the level 1 headers must be expanded etc. The parameters belong to the nearest header above it; it is thus not possible to construct a tree looking like this:

Header 1
Header 2
Header 3
-parameter 1
-parameter 2

The application will have full functionality even if no headers is used, then all parameters simply will be displayed at the first level.

The parameter lines have a predefined format that must be followed. As previously mentioned the parameter lines must start with a "-", the format for a parameter line looks like

```
-parameterName:DefaultValue:ValueRestrictions:ParameterType
```

`parameterName`	the name of the parameter.
`DefaultValue`	is the default value of the specific parameter. This field has simply an informational purpose, and does not affect the values in the template file (the file describing the run for the docking program).
`ValueRestrictions`	this field describes the possible values a parameter can take.
`ParameterType`	In this field the type of the parameter must be stated. The application can handle six typed of parameters: file, string, double, integer, Boolean and float.

The two fields above that are necessary for each parameter line is the parameter name and parameter type field. If a field is excluded the colons must be left, an example:

The parameter *filename* has not got any default value and no value restrictions, the resulting parameter line looks like

```
-fileName:::file
```

Is it preferable to not use any spaces in a parameter line.

The default value is only meant as information for the user and the value entered here should be checked against the used template file so as not to give the user incorrect information.

The application can handle six types of parameters:

string	is a sequence of characters
file	is a file, the application can not control the specific type of file given to the application, the responsibility for choosing the correct type of file lies with the user
boolean	a parameter taking the values true or false, this type can either be written as bool or Boolean
integer	a integer parameter, can be written as integer or int
float	a decimal number, smaller than double
double	a decimal number, bigger than float

The two types float and double differs in the place they take in the memory of the computer. If not explicitly stated what kind of decimal number to use, use double.

The value restrictions field is checked for parameters of the type string, integer, float and double. The field can contain either a number of strings or one of the following expressions (>X, <X, X-Y), where X and Y corresponds to numbers, float/decimal number for parameters of the type float or double and integers for the parameters of type integer. The expressions correspond to

>X	parameter should be bigger than X
<X	parameter should be smaller than X
X-Y	parameter should be smaller than Y, bigger than X or equal to one of them

When the parameter is of string type, the field should either be empty (interpreted by the application as this parameter only can take one, arbitrary, value) or containing the values the parameter can take, separated by a comma according to

```
parameterName:DefaultValue:PossibleString1,PossibleString2,.
..:ParameterType
```

If the parameter *parameter2* is a string and can take the values *Value1*, *Value2* and *Value3*, and the default value of the parameter is *Value1* the corresponding parameter line looks like

```
-parameter2:Value1:Value1,Value2,Value3:string
```

Appendix C: User Manual

C-1 To start the program

To start the program the .jar-file called DockControl.jar is needed. Go to the directory in which the file is and write *java -jar DockControl.jar* in the terminal window. Now the program starts.

C-2 General information about the scripts

The application executes both scripts written in perl or in python. If entering another type of script in any of the "script text areas" an error message will appear.

C-3 To make the settings

The application has three parts, the start, the settings and the run, each of these parts has its own graphical user interface (GUI). In the first two GUI:s the user makes settings, in the last the application execute the script/s corresponding to the settings.

When the program is started a window appears (figure C-1). In this window the name of the docking program and the localization of the parameter file are entered. Please remember that when entering information manually in a text area, the information is not submitted until the user presses enter. For more information about the parameter file please see Appendix B.

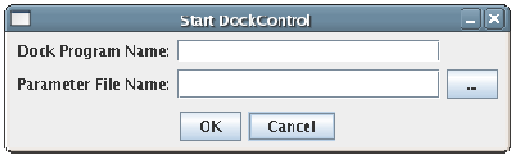


Figure C-1. The dialog displayed upon starting the docking.

C-3.1 The program settings

Once the program name and the parameter file have been entered a new interface is opened (figure C-2). The interface has two parts; in the upper half the input to the script running the docking program is given. In the lower part the design settings is chosen.

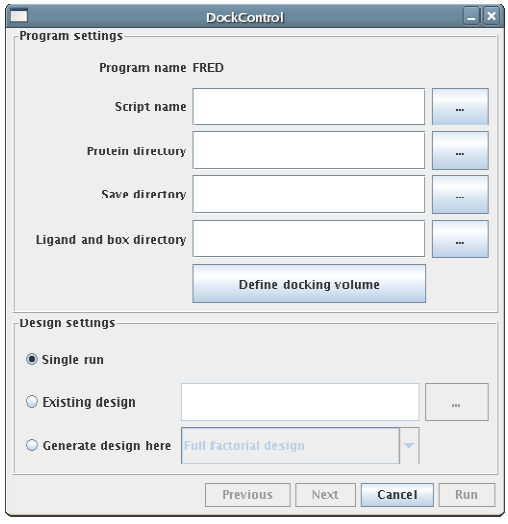


Figure C-2. The first view of the DockControl GUI, in the text called the program settings view.

In the upper half the following information can be entered:

1. The name of the script running the docking program.
2. The name of the directory containing the protein files.
3. The name of the directory in which all information will be saved.
4. The name of the directory containing the ligand and box files.

In the lower part of the interface the design information is entered. The user can choose between making just one run of the application, using an already constructed design or using the application to generate the design.

In the first case, the application will be used to enter the chosen settings for the parameter at that single run. In the second case the name of the design file must be entered, and in the third case the user chooses if a full factorial or fractional factorial design will be made.

When using the application for constructing a design the next view will be the design view, when submitting a design file the next view is the one in which post treatment of data is chosen.

C-3.1.1 Using the application to produce the box files

There is a possibility to producing the box files at the same time as the directory containing them is specified. To do this just press the button called “Define docking volume”, now a new window opens (figure C-3), in this the script producing boxes, the directory containing the crystal files and the directory in which the boxes will be saved is entered. Thereafter the button “Run script” is pressed. When the execution of this script has ended the name of the directory containing the box files is automatically entered into the appropriate text area.

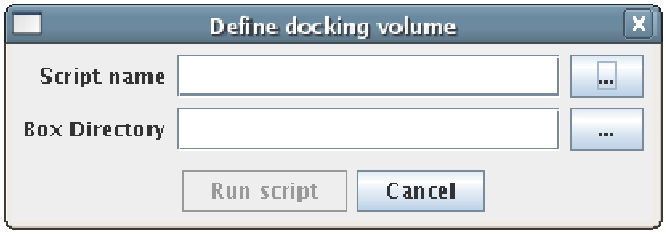


Figure C-3. The active site dialog.

C-3.2 The design settings

The upper part of this interface varies in appearance depending on what design the user wants to make, the reason for this is that the upper part of the interface displays information about the chosen design. The lower part of the interface, however, is always the same.

When pressing the “Choose parameters” button a window appears with a tree displaying the parameters for the docking program, for more information about the layout of the tree, please see Appendix A. To choose a parameter double click on that parameter, now a dialog in which the parameter settings can be made appears. The dialogs differ a bit between different kind of parameters and it is always possible to see at the top of the window what kind of values the parameter can take.

When the settings are chosen the parameter appears in the table of the interface. If the user chooses a parameter for which the settings are already made, the application notifies the user about this and asks whether the settings for the parameter in question should be updated.

To remove a chosen parameter, highlight the parameter row and press the “Remove” button.

When enough parameters is chosen the “Make design” button is enabled, this button displays a window showing the finished design with triplicate center points (except for when doing a design for a single run). In this window it is possible to choose between saving and discarding the design. To be able to continue to the interface in which the settings for the post treatment of data is chosen a design must be saved.

C-3.2.1 How to make a design for a single run

The design interface when making a single run is depicted in figure C-4. Here two information fields is displayed and an option for the user to make a default run of the docking program.

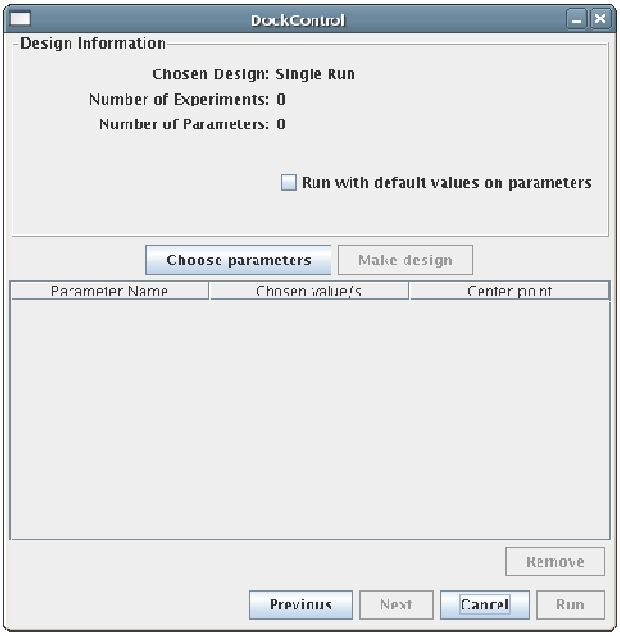


Figure C-4. The design interface when making a single run.

The information fields depicted here is the number of experiments and the number of parameters chosen. These fields are updated every time the user makes a change in the number of parameters chosen for the current run. The dialogs for choosing parameter settings is quite simple, the user can only choose one value for each parameter.

C-3.2.2 How to make a default run of the docking program

To make a default run of the docking program there script that will execute the docking program must fulfil some requirements, these requirements can be found in Appendix A.

C- 3.2.3 How to make a full factorial design

The design interface when making a full factorial run is depicted in figure C-5. The information shown here is the number of experiments resulting from the number of chosen parameters and the number of parameters chosen.

The dialogs for choosing parameter settings for a full factorial design is somewhat more complicated than the dialogs for setting a single run parameter. The full factorial design takes any number of values for a given parameter. This application, however, accepts only several values for number parameter (i.e. integer or decimal parameters), boolean parameters or parameters with predefined values for its settings. In the case of number parameters a lower and upper limit is chosen then the application calculates a chosen number of values evenly between these limits. Another difference between a single run design and a full factorial

design is that a full factorial design contains triplicate center points. The values for these points must be chosen if the parameter has a number of predefined string values, if the parameter is a number parameter the center point is calculated.

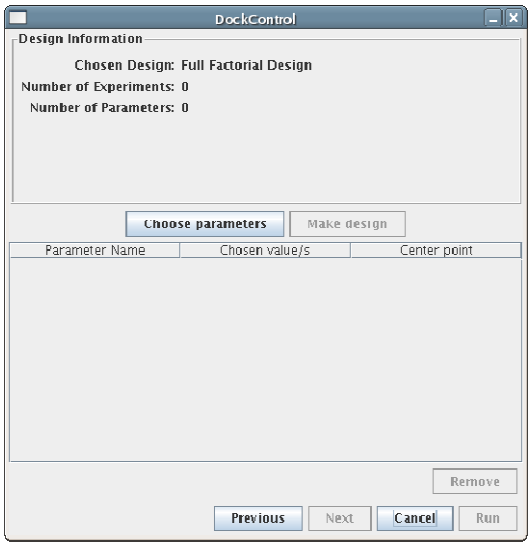


Figure C-5. The design interface when making a full factorial design.

C-3.2.4 How to make a fractional factorial design

This is the design interface containing the most information (figure C-6). The information shown is the number of experiments resulting from the number of chosen parameters, the total number of parameters chosen, the number of parameter that is varied in the design, the number of levels this fractional design has and the number of levels (that is the number of values a specific parameter can have, the center point values not counted). In the interface it is also possible to choose the reduction factor, the number used to decide how many experiments the fractional factorial design will consist of according to $2^{\text{No. changing parameters} - \text{reduction factor}}$. From the chosen reduction factor the resolution of the design is derived, this number is also shown in the design interface.

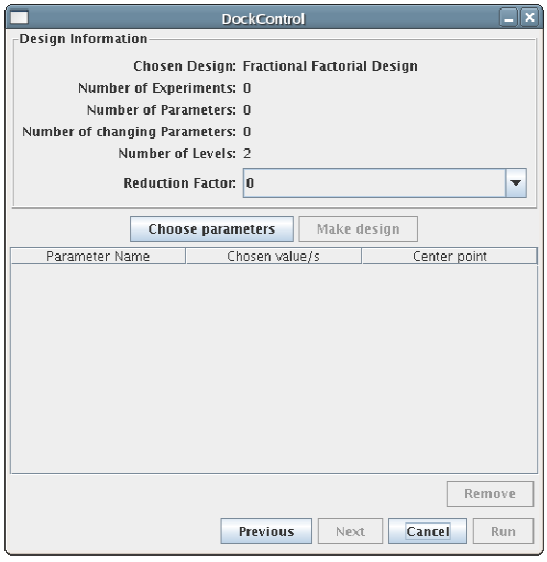


Figure C-6. The design interface when making a fractional factorial design.

The dialogs for making a full factorial design functions as for the full factorial design except for that the parameters in a full factorial design maximally can have two values. The value for the center point still has to be chosen.

C-3.3 The output evaluation settings

In this interface the output evaluation can be chosen. The interface (figure C-7) is fairly straightforward in that the user chooses the scripts that will be run and then chooses the input to these scripts. All chosen scripts will also receive as input the directory in which the output of the docking program was saved.

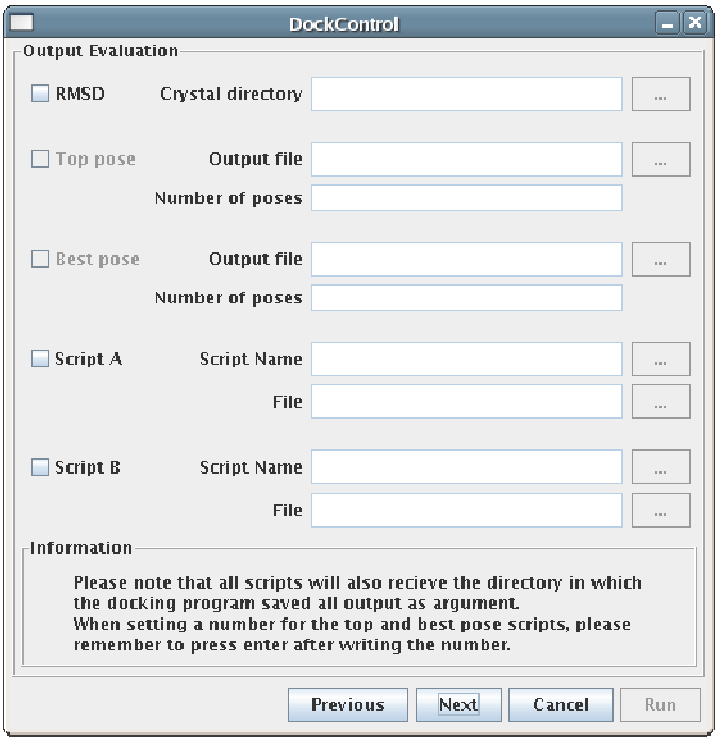


Figure C-7. The panel in which the treatment of the data resulting from the docking run/s can be chosen.

For the application to be able to run the RMSD, top pose and best pose script these scripts must be found in a directory called scripts in the same directory as the DockControl jar file. The scripts must be called rmsdscript, topscript and bestscript, respectively, for more information, please see Appendix A.

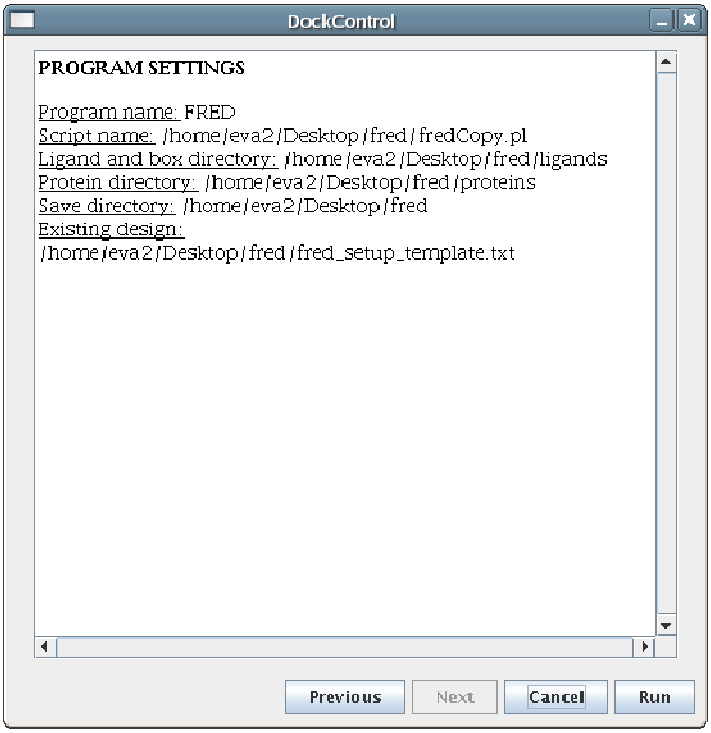


Figure C-8. The settings chosen by the user displayed.

When the settings are made in this interface, the next view (figure C-8) shows the chosen settings. In this view it is possible to press the "Run" button whereupon the window displaying output from the chosen scripts (figure C-9) appears. When the execution is finished a dialog informing the user that the run is finished appears.

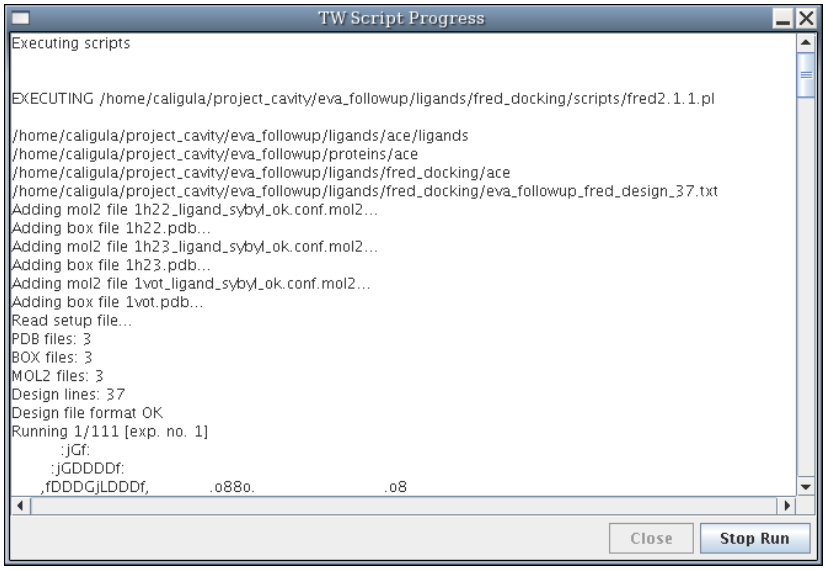
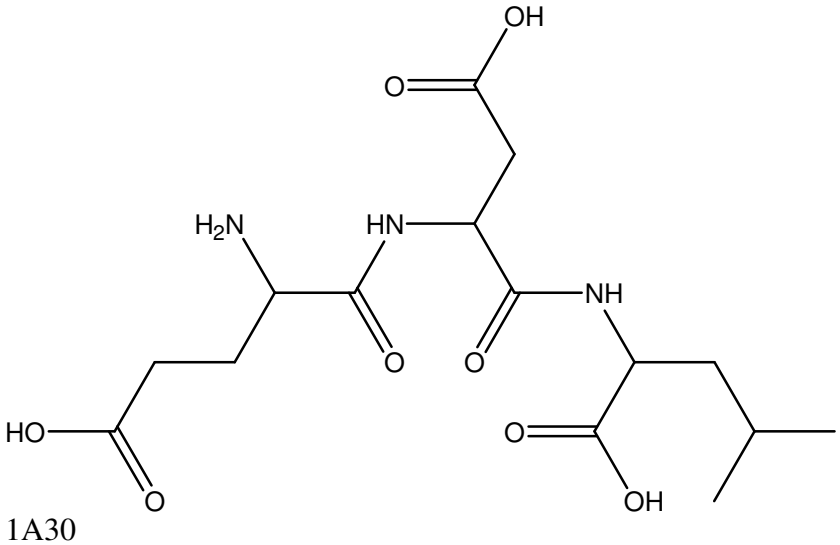


Figure C-9. The display of the script execution progress.

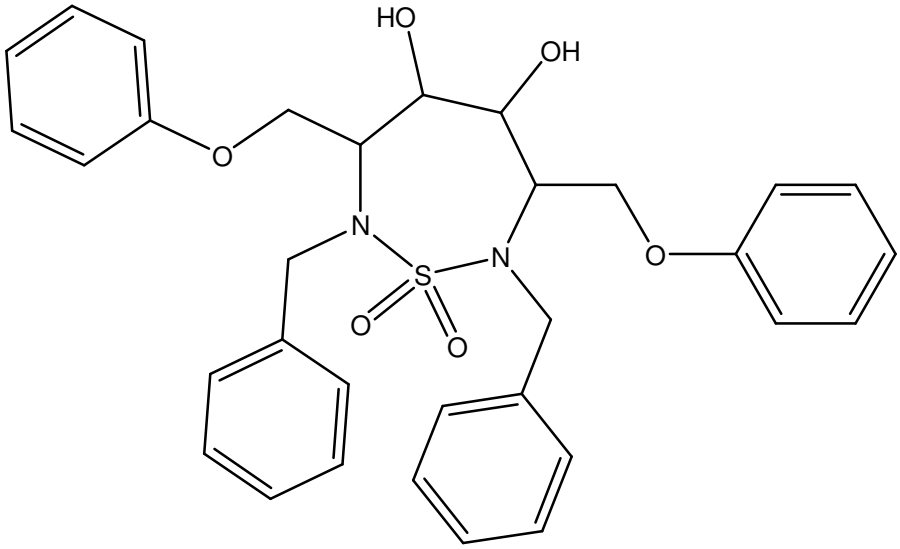
C-4 Requirements

The user needs the .jar file called DockControl.jar to be able to run the application. Scripts for controlling the docking program of interest (see Appendix A) and a parameter file containing information about the parameters of the docking program (see Appendix B) is also needed. To be able to fully make use of the application, scripts for calculating RMSD and extracting top and best pose at the specified location (Appendix A) is also needed.

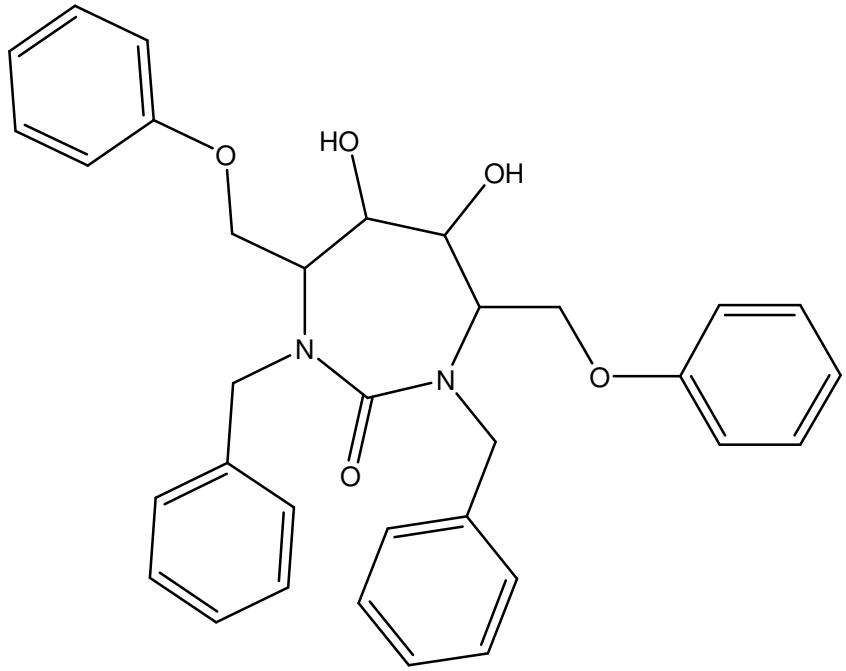
Appendix D – The docked ligands



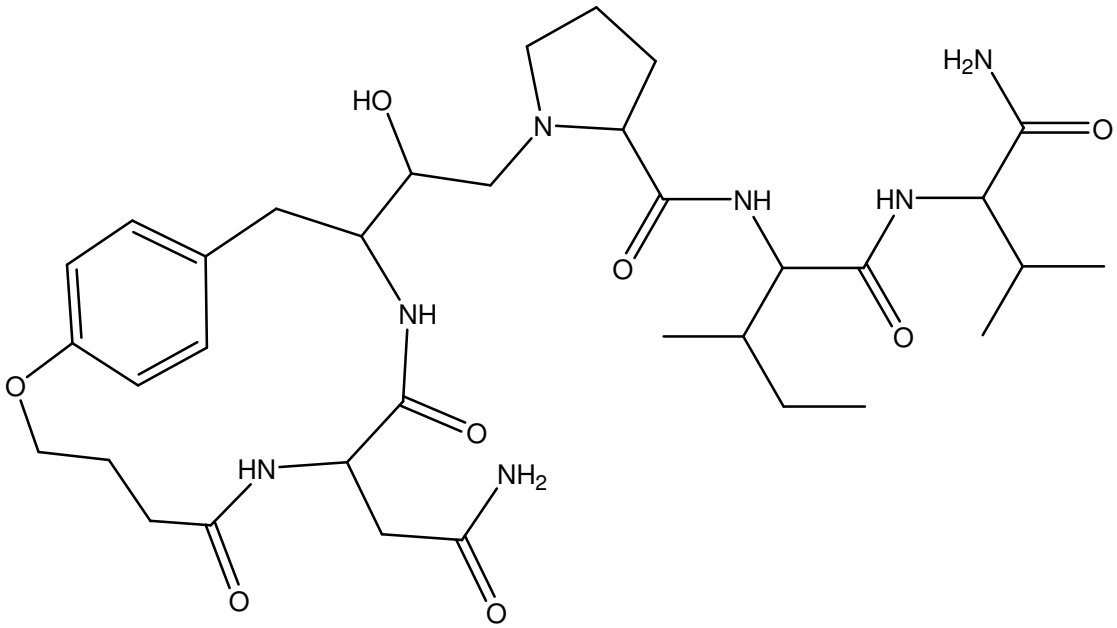
1A30



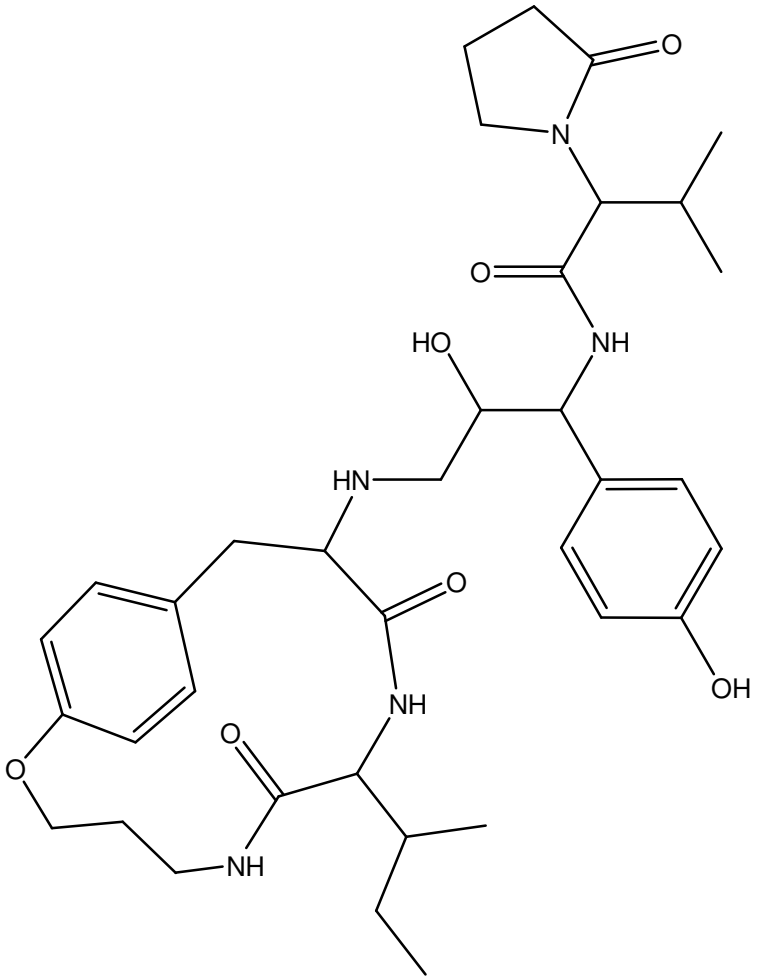
1AJV



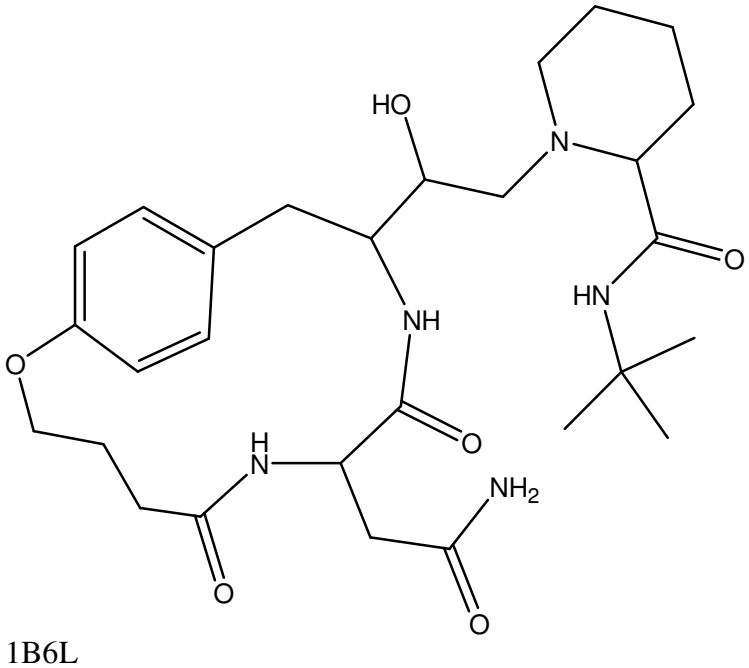
1AJX



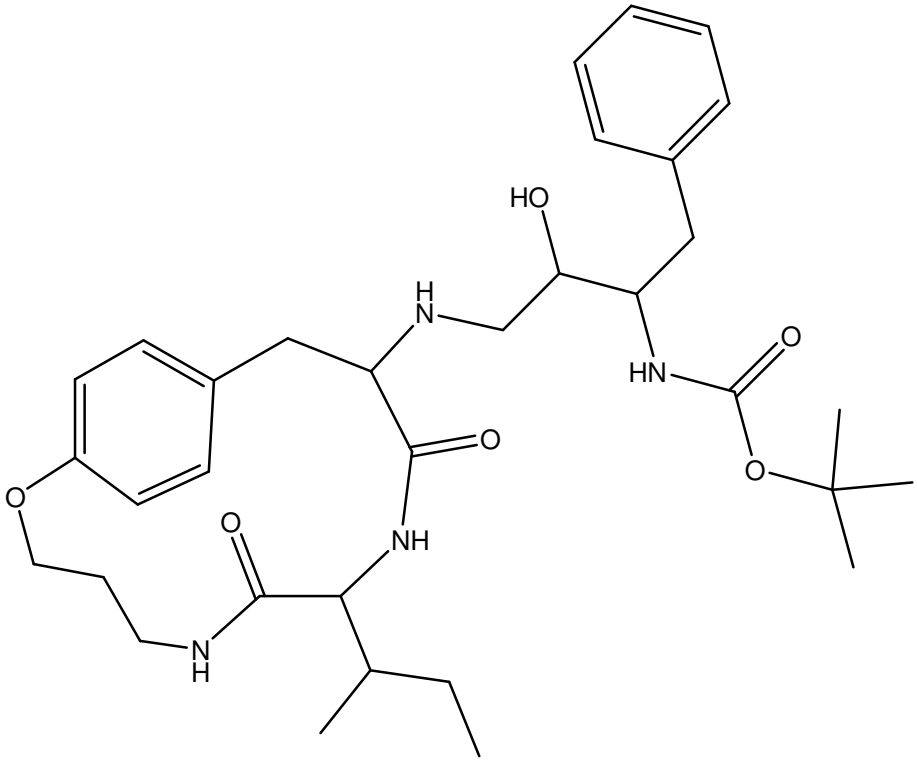
1B6J



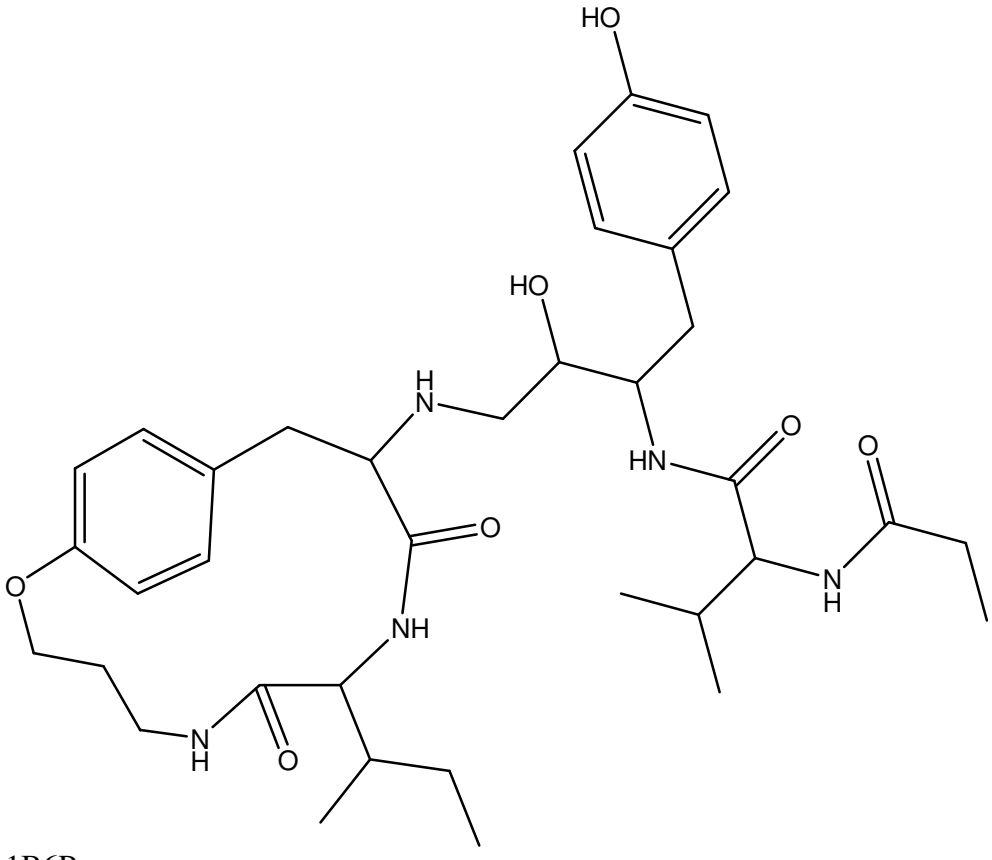
1B6K



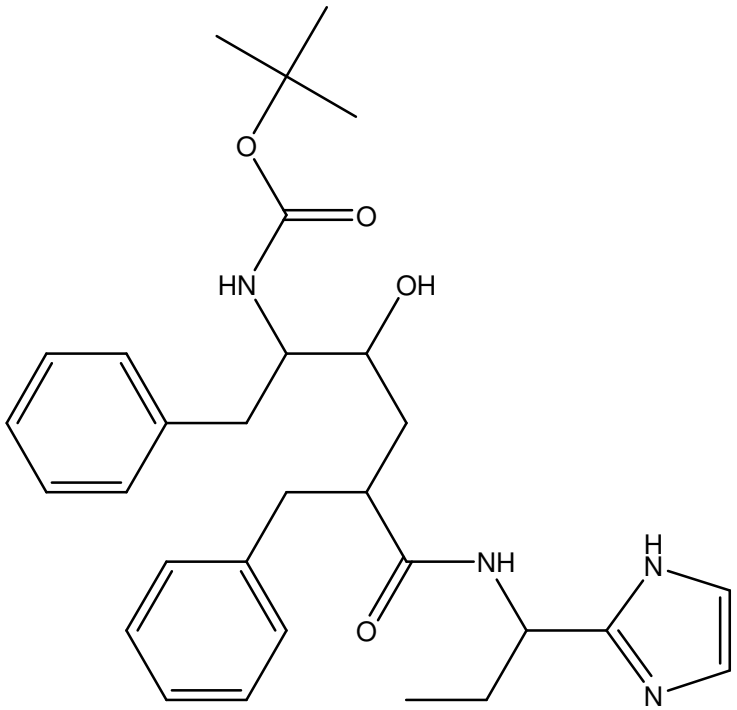
1B6L



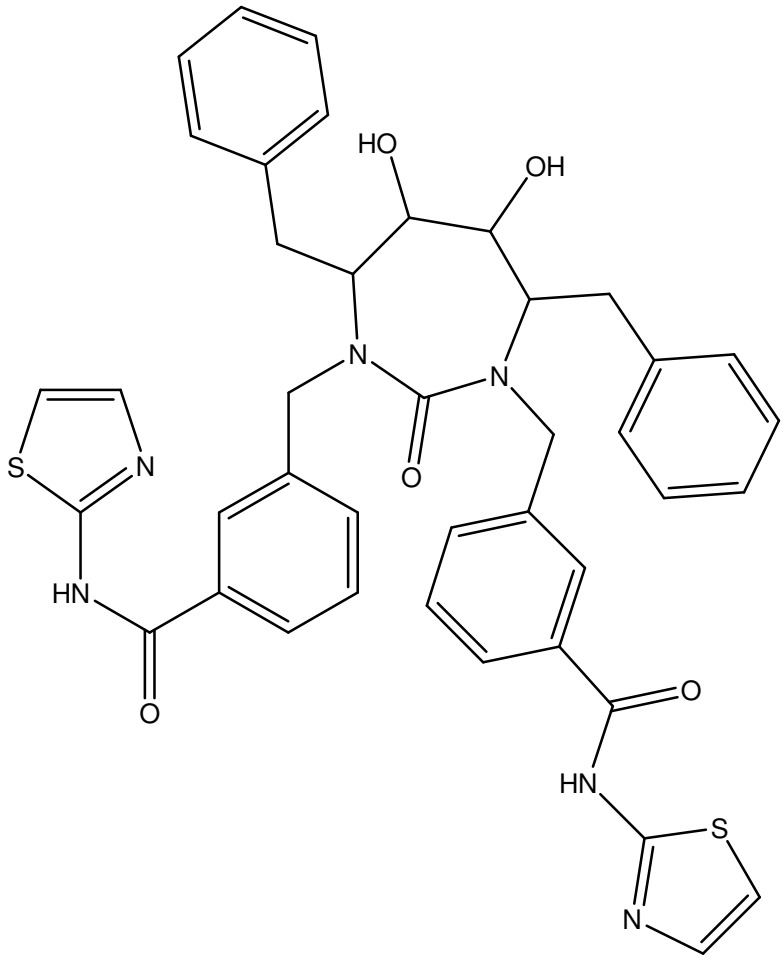
1B6M



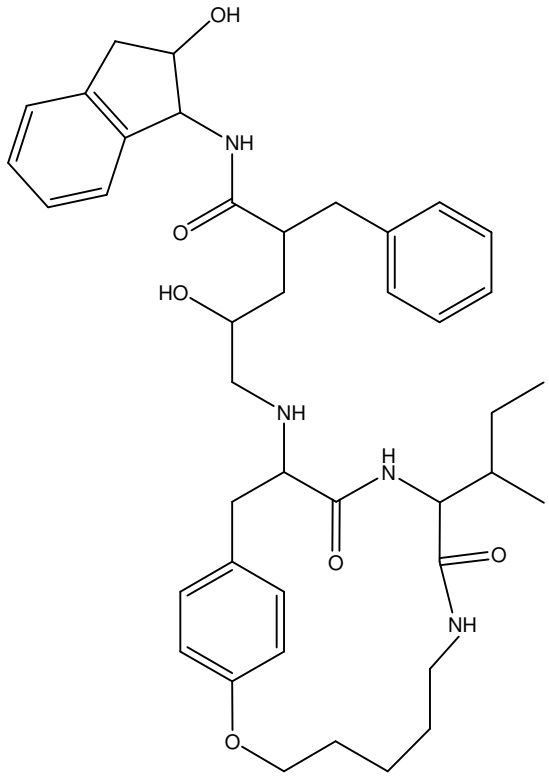
1B6P



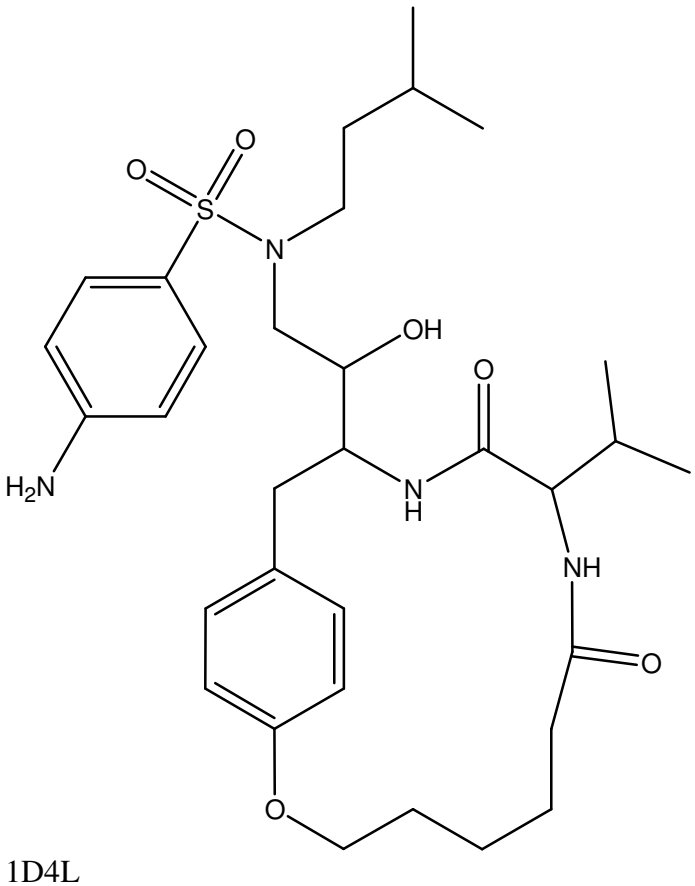
1BDQ



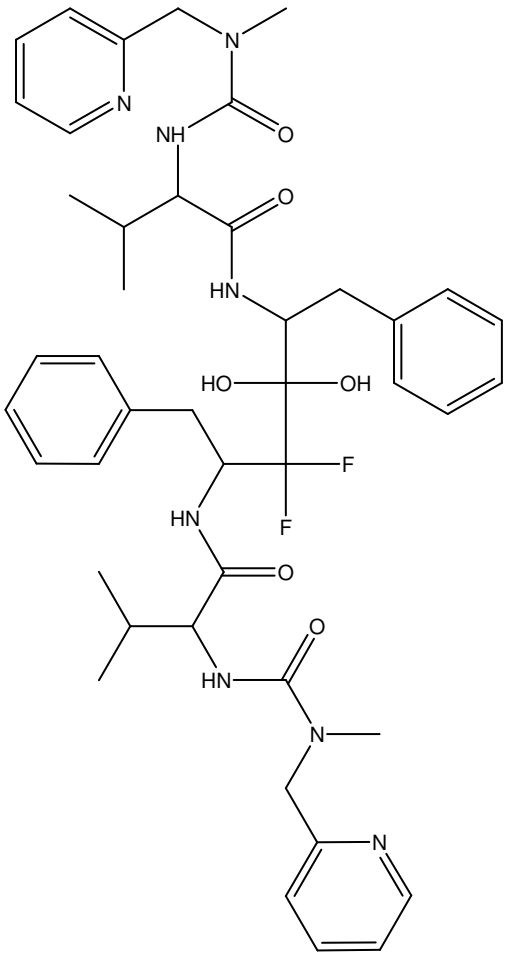
1BV9



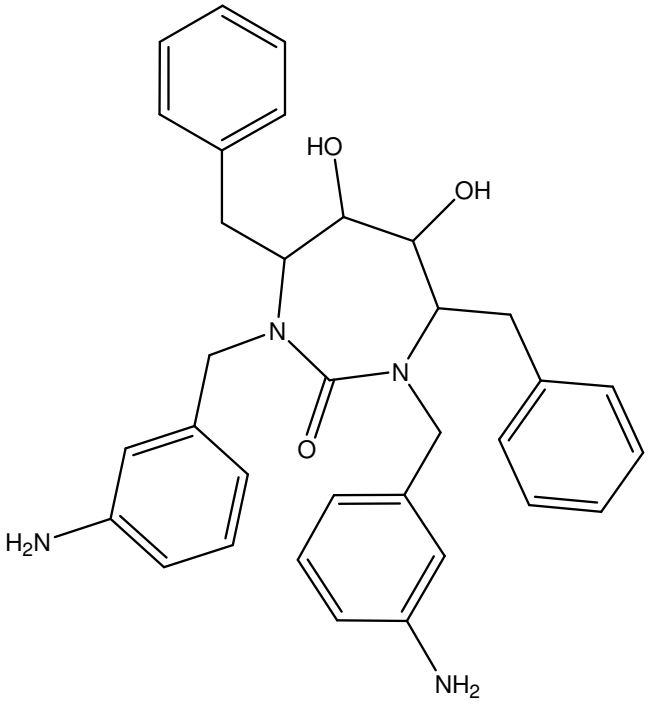
1D4K



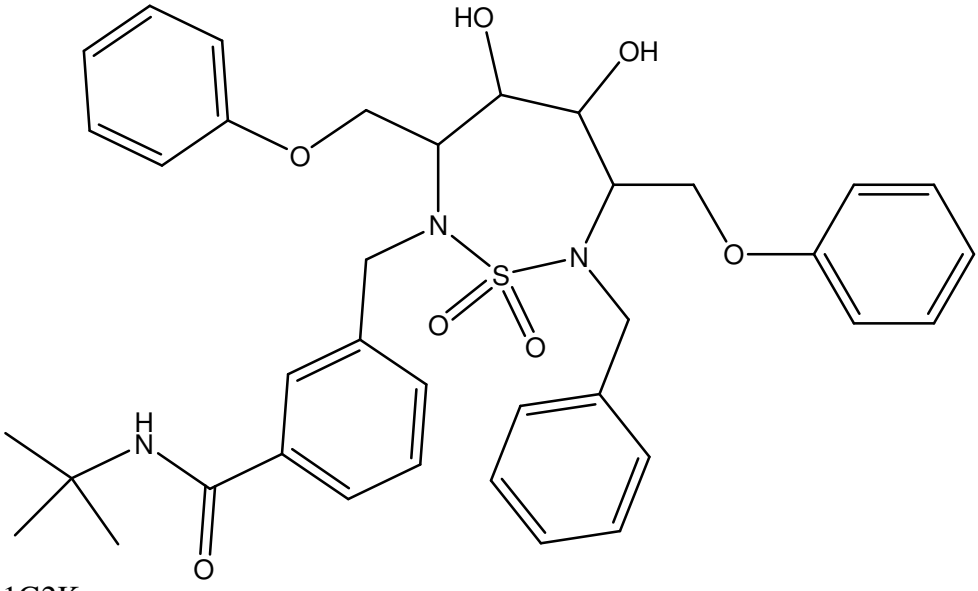
1D4L



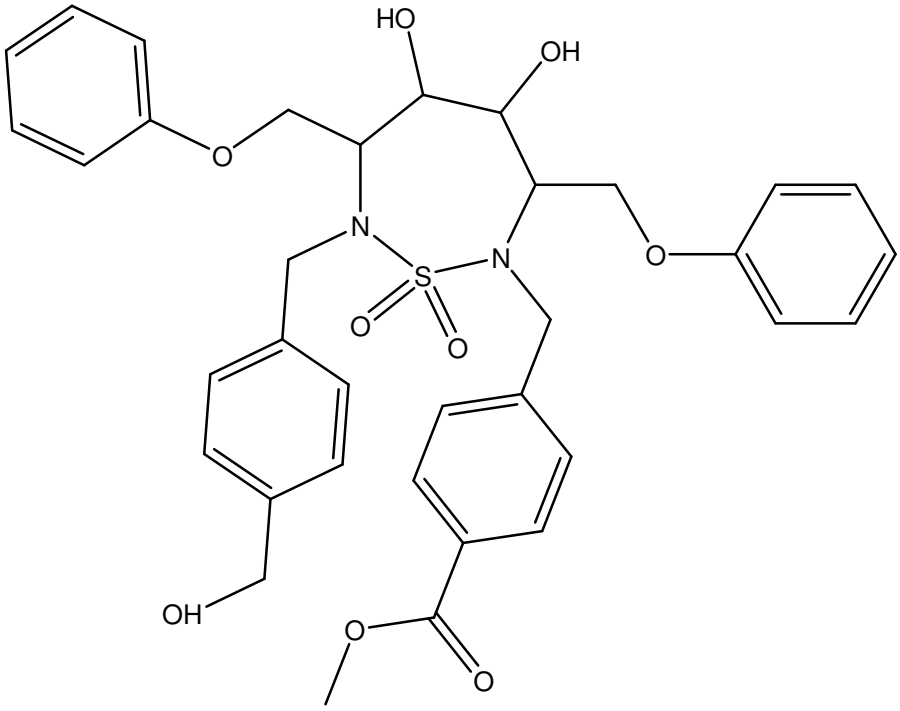
1DIF



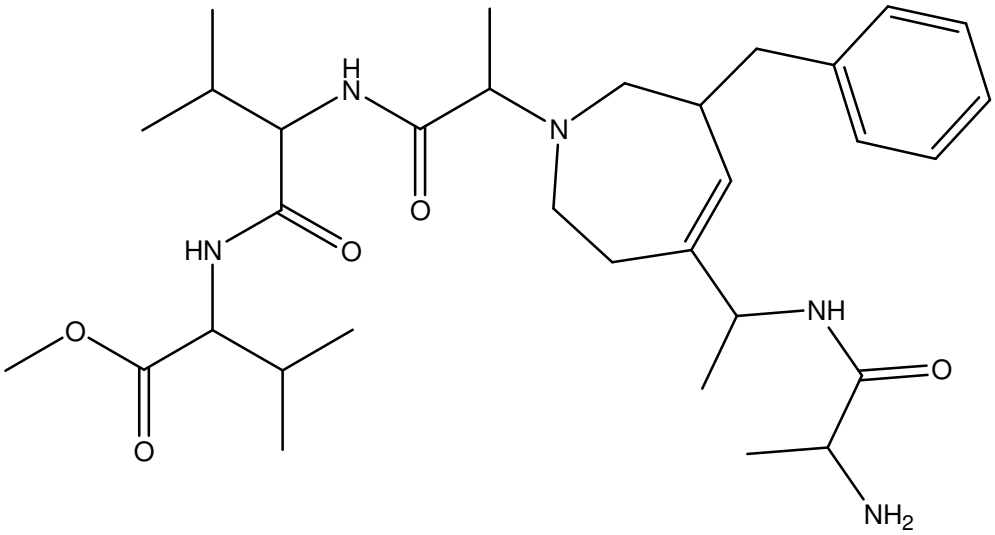
1DMP



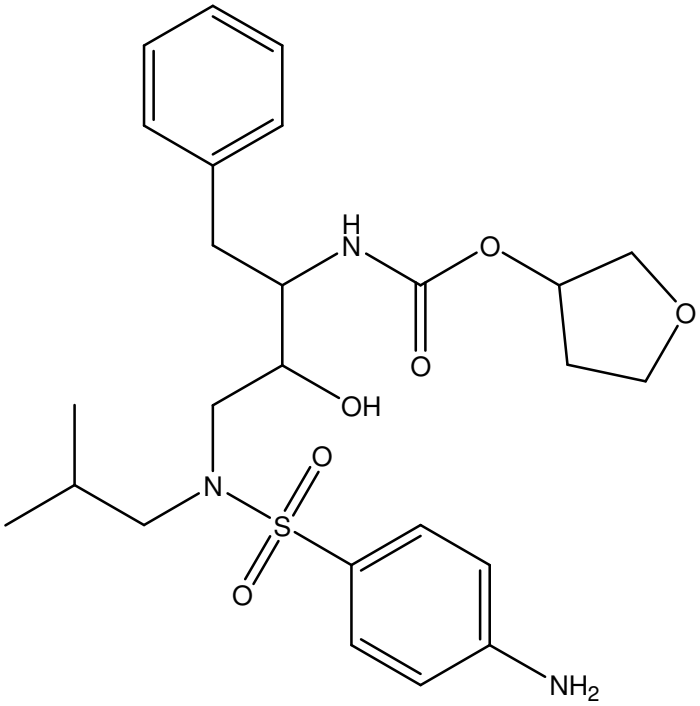
1G2K



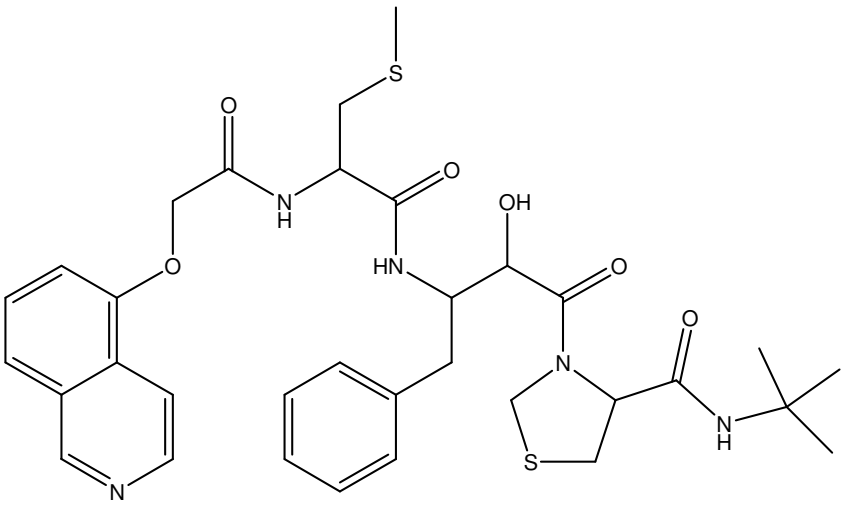
1G35



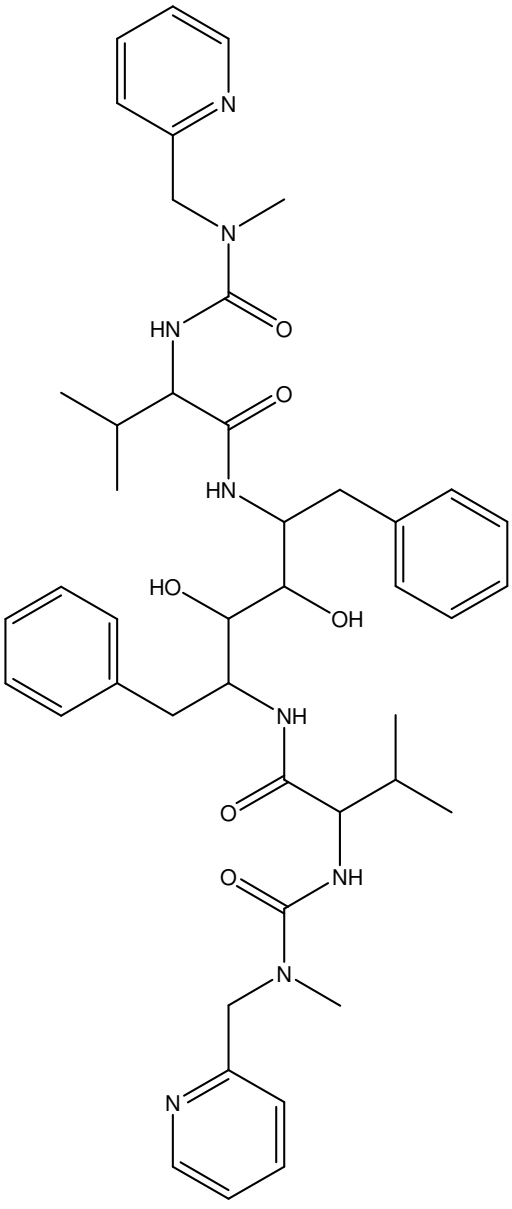
1HBV



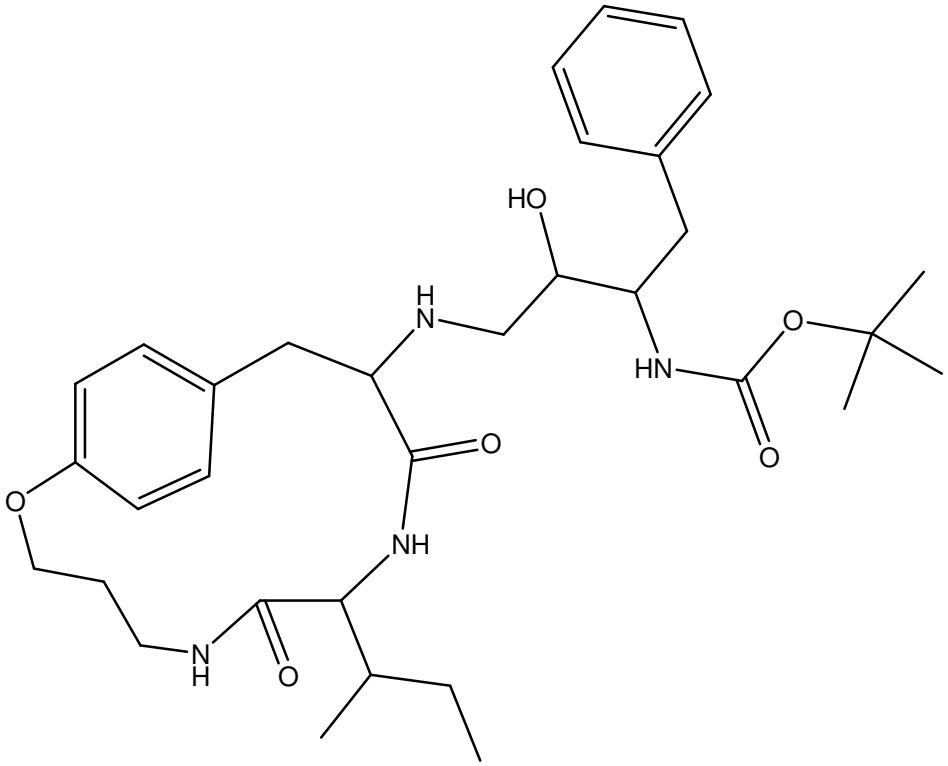
1HPV



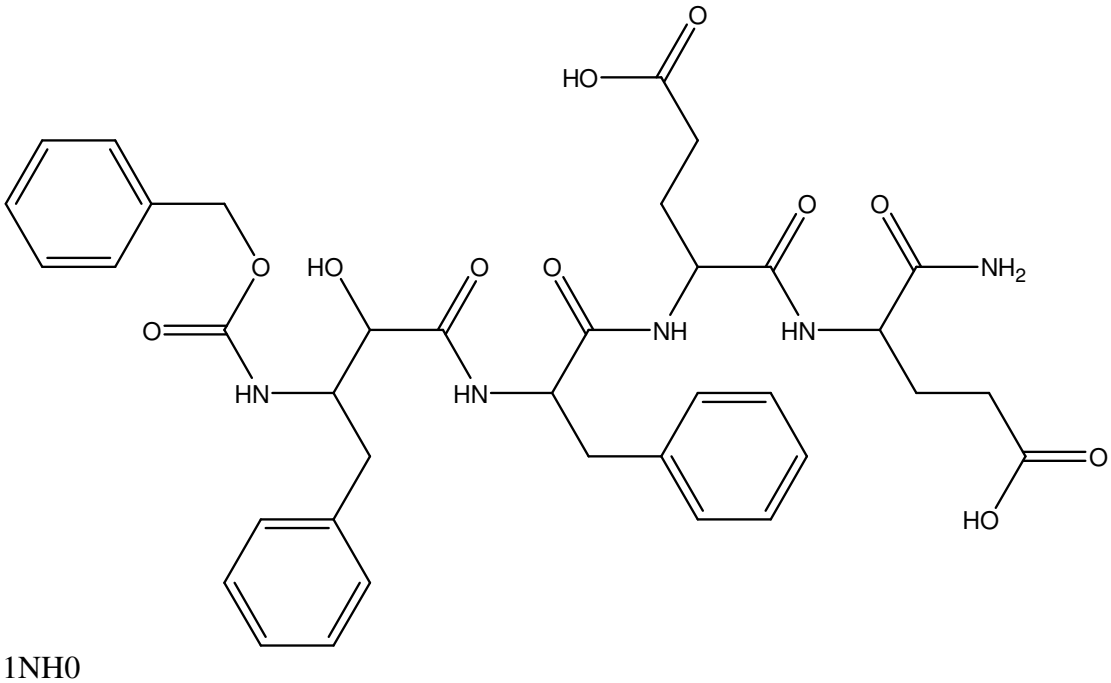
1HPX



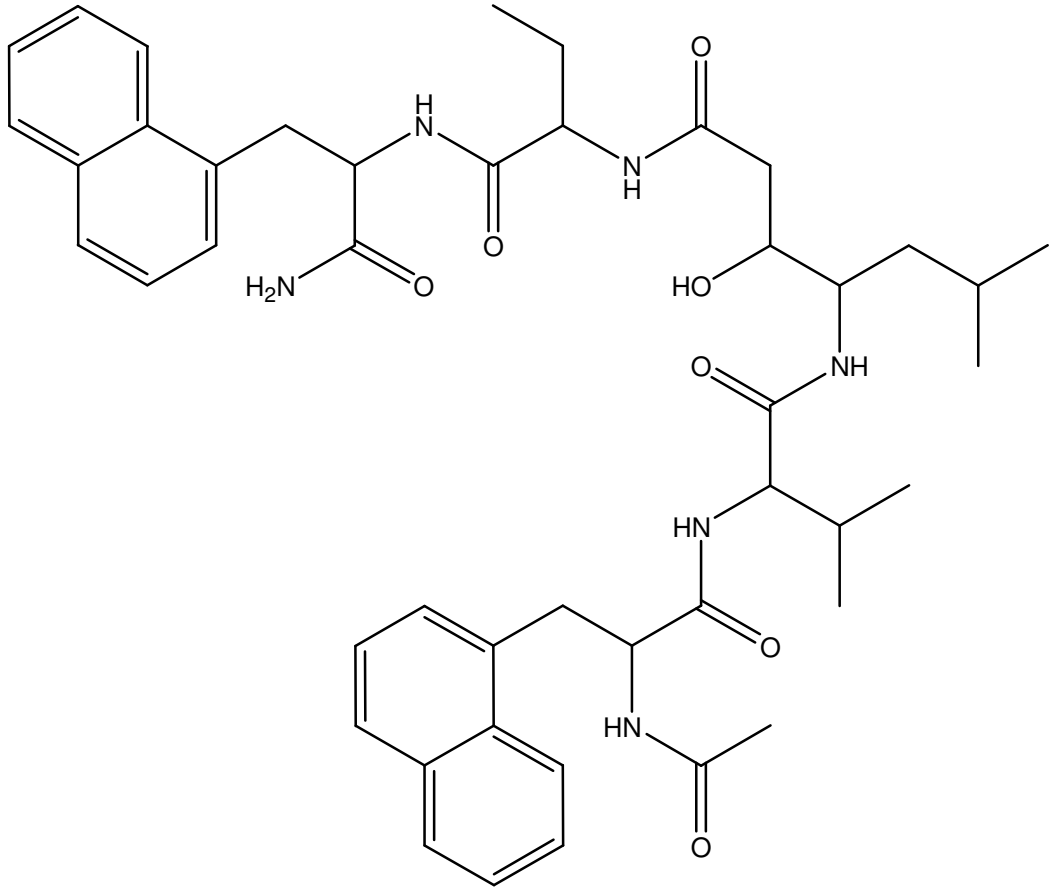
1HVI



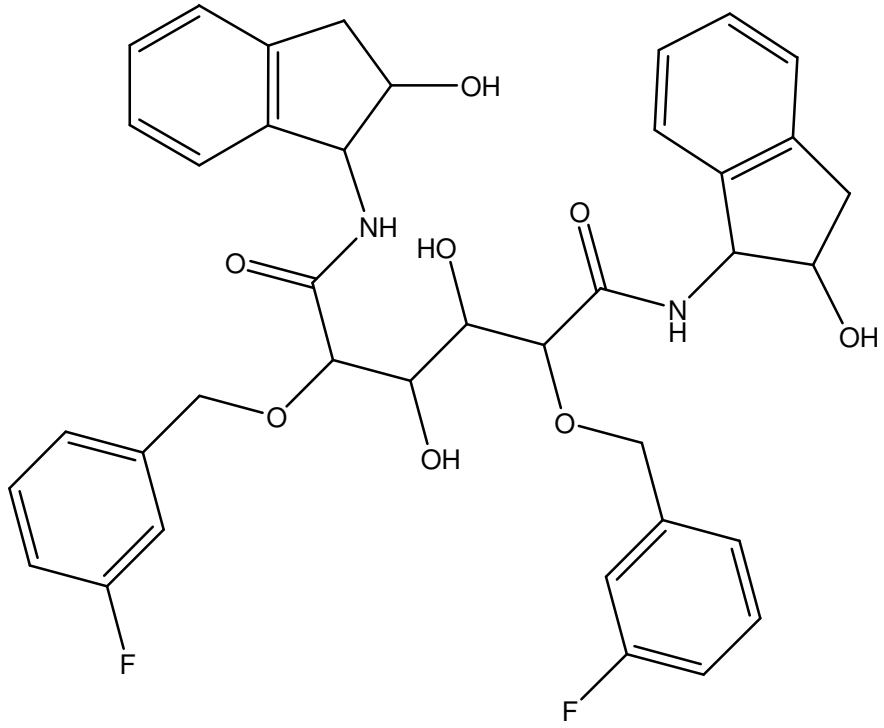
1MTR



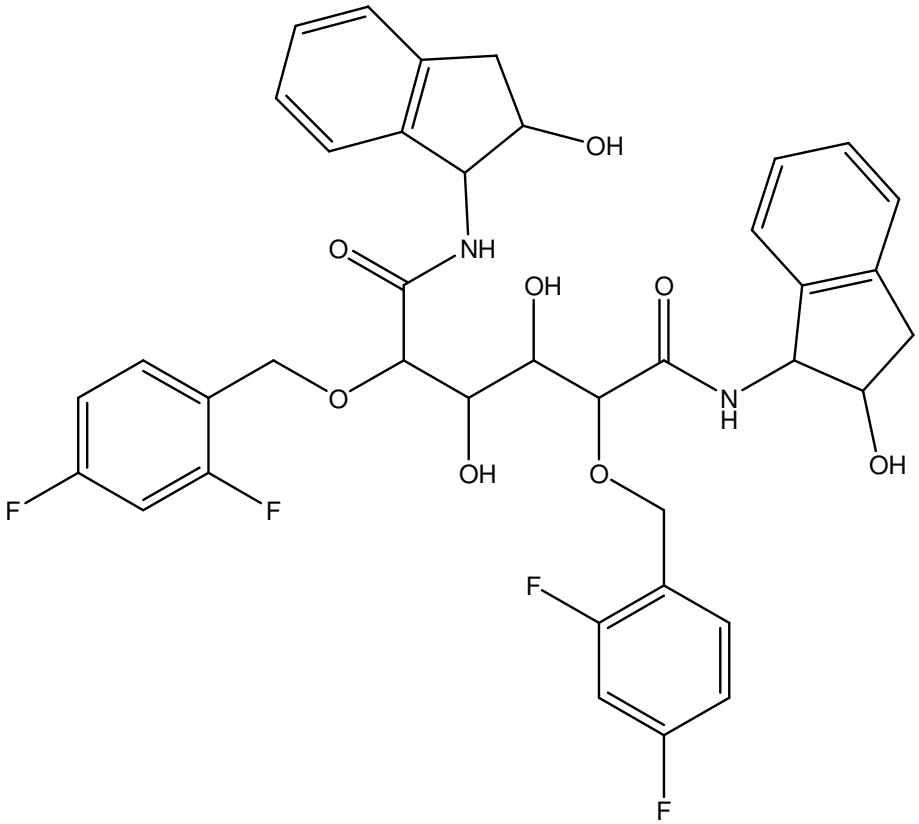
1NH0



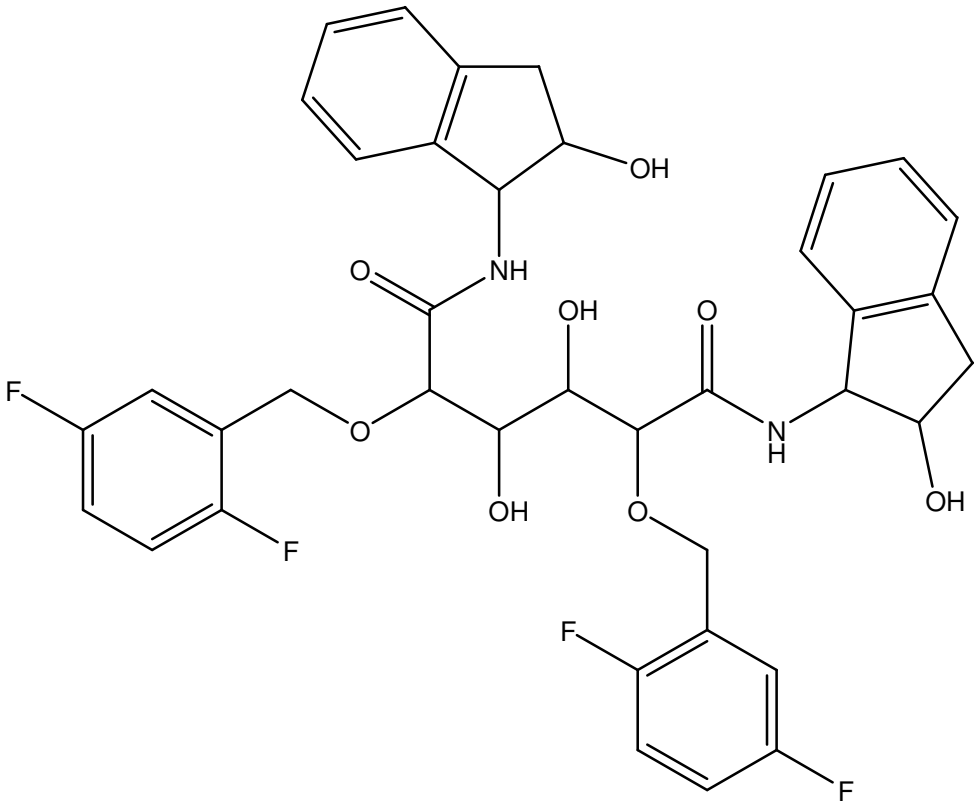
1ODY



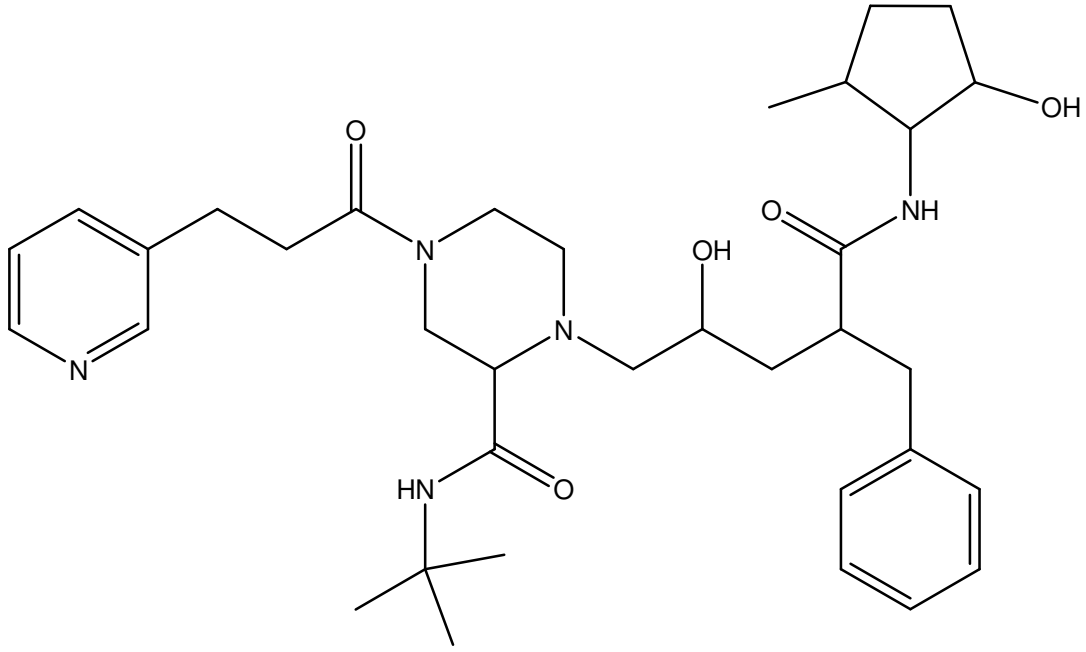
1W5V



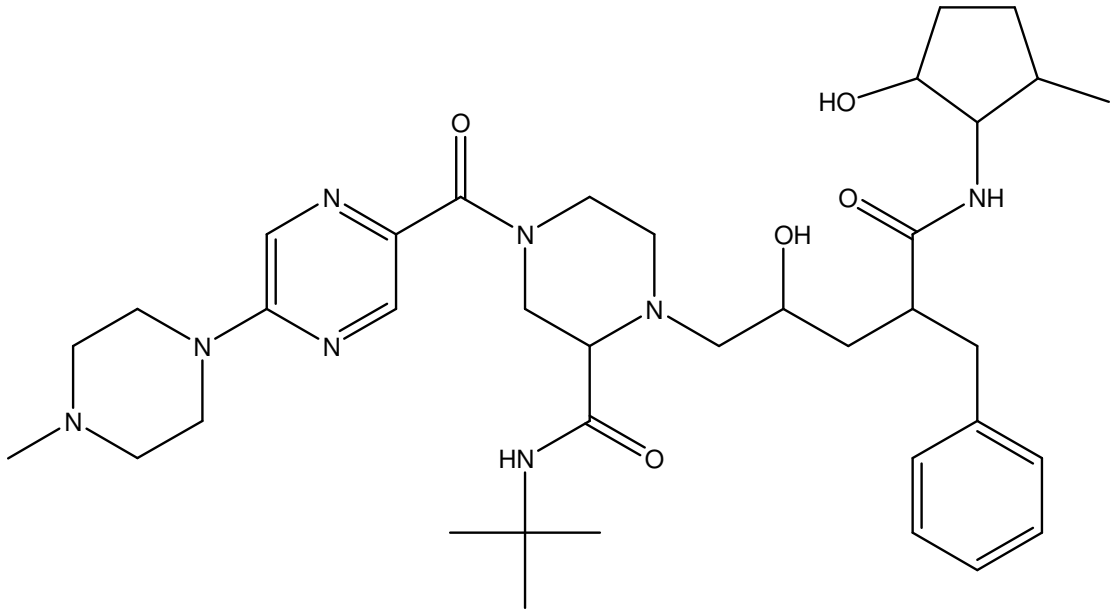
1W5W



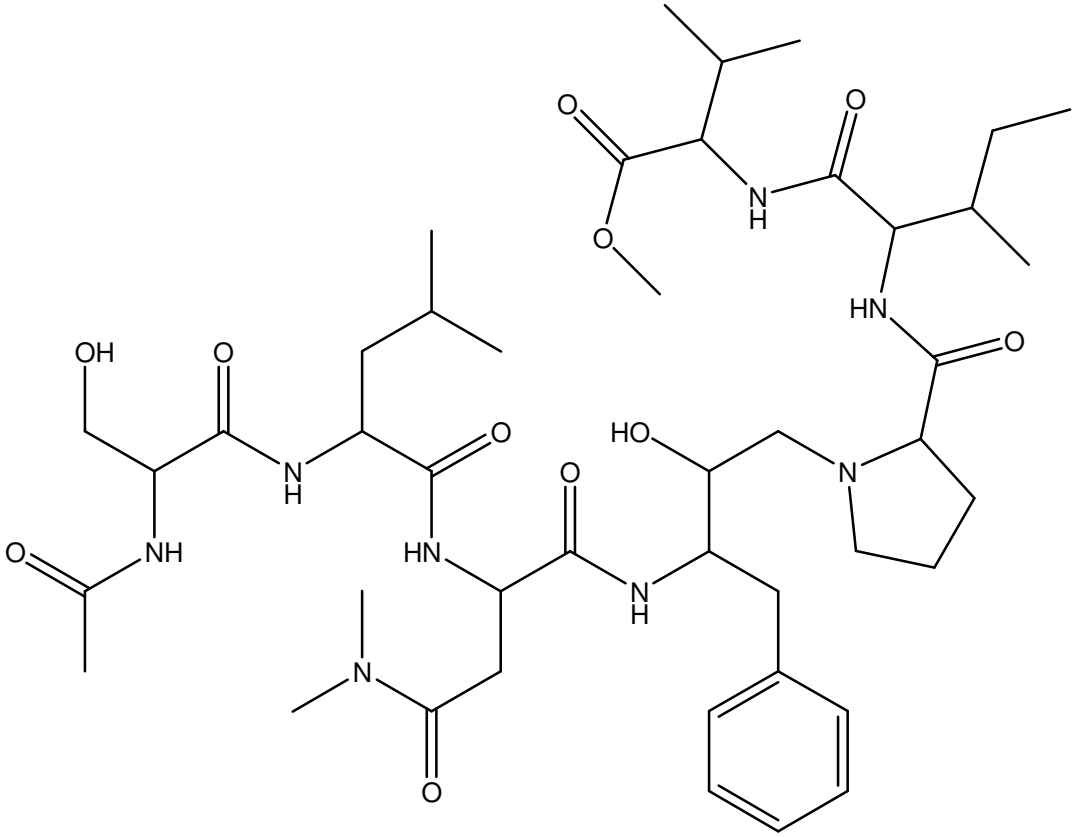
1W5Y



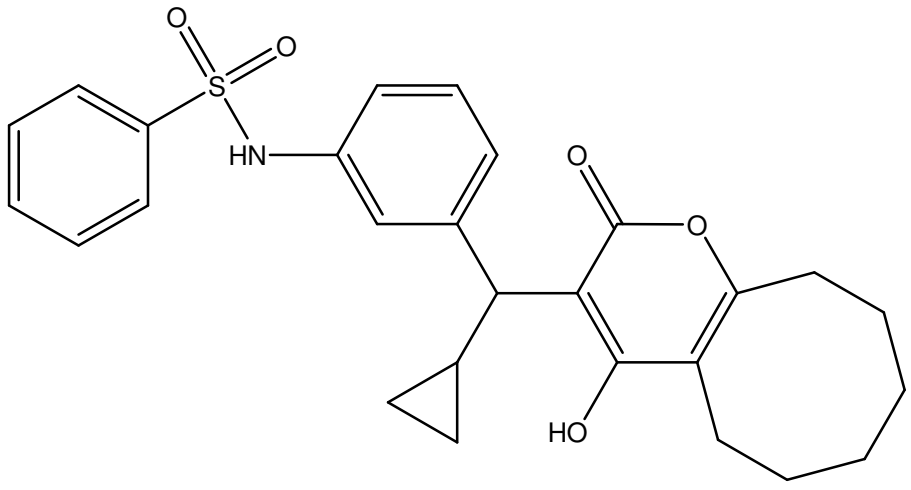
2BPV



2BPY



7HVP



7UPJ