Searching for Similar Binding Sites

Master thesis

Anders Thøgersen

July 14, 2009

Abstract

In protein research it is often important to determine with which other proteins an interaction may occur. This can help in definition of biological role of the protein, such as inhibition or enhancement of particular functions. The criteria that makes an interaction possible is that the two proteins contain *binding sites* that structurally and chemically allow the interaction to take place.

In this work a method is developed to allow the deduction of possible new protein interactions by looking at the set of protein interactions that are already known.

A framework has been created for experimenting with superimposition of protein binding sites. The central methods employed are *Particle Swarm Optimization* and *Iterative Closest Point*. Because the same protein binding site may have variations across protein families, the removal of *outliers*, i.e. atoms that should be discarded to achieve a good superimposition of the binding site core, has been a central theme of this work.

The developed methodology is calibrated on representative data and the quality of superimpositions made is compared with *MultiBind*, an existing program for solving this problem that uses a fundamentally different approach.

Good superimpositions are achieved with geometrically similar structures, but biologically significant results are obtained less frequently. However, some experiments indicate that there is room for significantly improving performance on biologically significant results.

Resumé

I proteinforskning er det ofte vigtigt at kunne bestemme med hvilke andre proteiner en interaktion kan finde sted. Dette kan hjælpe med at definere et proteins biologiske rolle som inhibitor eller fremmer af bestemte funktioner. Det der gør interaktion mellem to proteiner mulig er at de indeholder en region der strukturelt og kemisk tillader interaktionen at finde sted.

I dette arbejde udvikles en metode der tillader deduktion af mulige nye interaktioner mellem proteiner ved at se på det sættet af allerede kendte protein interaktioner.

Programmel er blevet udviklet som muliggør eksperimenteren med samlægning af proteiners bindings regioner. De centrale metoder der tages i brug er *Particle Swarm Optimization* og *Iterative Closest Point*. Fordi den samme bindingsregion i et protein kan have variationer i forskellige protein familier, har fjernelse af overflødige atomer i opnåelsen af en god samlægning af de centrale dele af en bindingsregion været et centralt tema i dette arbejde.

Den udviklede metode er blevet kalibreret på representativ data og kvaliteten af de opnåede samlægninger sammenlignes med *MultiBind* som er et eksisterende program til at løse det samme problem, men med en fundamentalt anderledes metode.

Gode samlægninger opnås med protein strukturer der ligner hinanden geometrisk, men biologisk signifikante resultater opnås mindre ofte. Dog indikerer nogle eksperimenter at der er plads til at forbedre ydelsen på biologisk signifikante resultater betydeligt.

Contents

This project has been done at the biotec center of Technische Universität Dresden at the structural bioinformatics group led by Dr. Maria Teresa Pisabarro in collaboration with M.Sc. Joan Teyra.

I would like to thank the group, especially Joan and Mayte for making this cooperation possible.

I would also like to thank my supervisor Jyrki Katajainen for good advice and support.

1 Background

Proteins are the most abundant biological macromolecules, occurring in all cells and all parts of cells. Proteins are the molecular instruments through which genetic information is expressed. Relatively simple monomeric subunits provide the key to the structure of the thousands of different proteins.

A protein is defined by both the backbone, which consists of a linked series of nitrogen, carbon and oxygen atoms, as well as the side chains which extend from the backbone and which are important for interactions between proteins. Carbon atoms that are part of the backbone are commonly referred to as Cα atoms.

All proteins are constructed from the same set of 20 amino acids, covalently linked in characteristic linear sequences. Because each of these amino acids have a side chain with distinctive chemical properties, this group of 20 precursor molecules may be regarded as the alphabet in which the language of protein structure is written. What is most remarkable is that cells can produce proteins with strikingly different properties and activities by joining the same 20 amino acids in many different combinations and sequences. The more general term *residue* is often used to denote the atoms that make up smaller chemical compounds, amino acids included.

The structure of proteins is typically divided into 4 different classes: *primary*, *secondary*, *tertiary* and *quaternary* structures.

The *primary* structure is the name used for the sequence of different amino acids that make up the protein. The sequence is determined by the gene corresponding to the protein.

Secondary structure refers to structures that have a regular geometry, such as the alpha-helices and beta-sheets. These structures depend only on properties of the proteins backbone, or main chain. Two common *Secondary structures* into which protein folds are the beta-sheets and alpha-helices which can be seen on figure 1.

The secondary structure elements are often folded into a shape of loops and turns referred to as the *tertiary* structure. Included are all the non-covalent bonds that are not a part of the secondary structure.

The *quaternary* structure denotes the interactions that happen between separate chains in the protein. Only proteins of a certain size consisting of several chains display quaternary structure.

The structure of a protein can give important clues about its properties and may be determined experimentally. Common methods for doing so include *X-ray crystallography* and *NMR spectroscopy*. Determining protein structure experimentally is a complex and involved process and only a fraction of structures of proteins is known. Known structures are available for download at the RCSB Protein Data Bank (PDB) [BWF$^+$00] which

Background

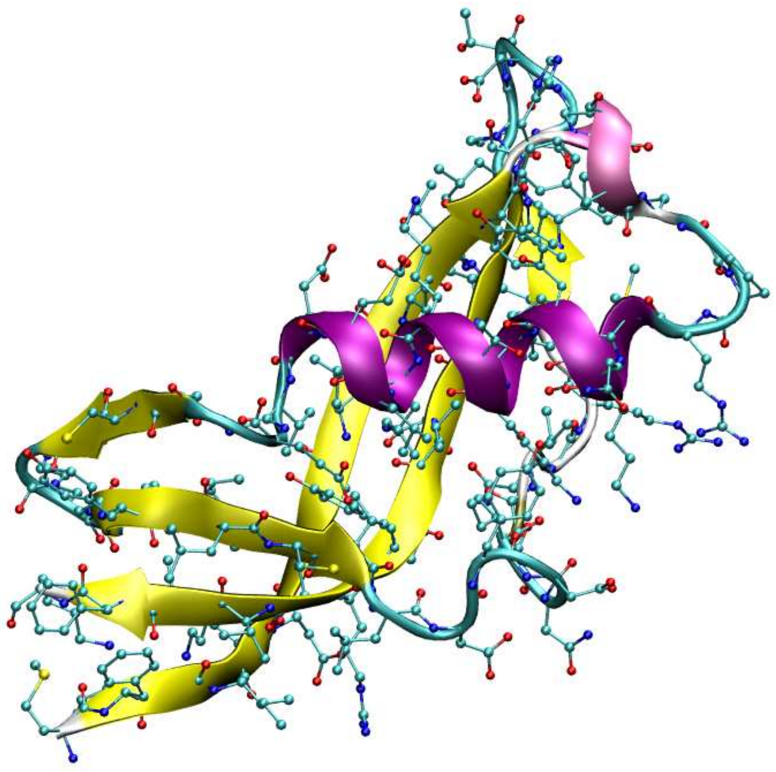


Figure 1: Stylistic view of a protein displaying the backbone and side chains. The backbone reflects the curve of the protein and consists of alpha-helices and beta-sheets, shown in Lila and yellow respectively. Numerous side-chains extend from the backbone and these are essential for interactions among proteins. The atoms making up the molecule are colored according to their element.

is the canonical archive of experimentally determined three-dimensional structures of biological macromolecules, including proteins and nucleic acids. At the time of this writing it contains around 55.000 determined protein structures.

The protein structures found in the PDB are distributed as flat files that contain a description of the atoms that make up the protein as well as additional information concerning physical and chemical properties of the structure and information about the measurement of the structure. The structures that may be found in the PDB files differ. Some files contain only a single protein while others contain several copies of the same structure. In other cases PDB files contain complexes formed between two protein molecules engaging in a binding.

In an attempt to generate a systematic overview of proteins, efforts have been made at classifying proteins into families, grouping proteins with similar structural characteristics together.

Most proteins fold into three dimensional structures. The folding of a protein into its native conformation is essential for the protein to function correctly. At the tertiary level, protein structures are seen as made up of more stable globular units called domains. The term domain denotes a part of the protein chain that can evolve and exist independently. Protein structure classification efforts typically rely use domains to group structures into families.

This approach is taken in the Structural Classification of Proteins (SCOP) [MCA+07] database. SCOP aims to provide a description of the structural and evolutionary relationships between known protein structures. It is created by manual inspection as well as use of automated methods.

1.1 Protein interaction

The functions of proteins rely on their interactions with other molecules including DNA, RNA, other proteins and other small molecules. Protein-protein interactions are especially interesting because they are found in virtually every process in living cells.

Two proteins interact by forming a non-covalent binding between their binding sites. A binding site is a definite place in the protein molecule which may bind a *ligand*. The term ligand designates a smaller molecule that is able to bind to a protein and thereby form new functionality. In our case the ligand is another protein. The protein that binds the ligand is typically referred to as the *receptor*.

A binding site is characterized by having a specific geometric shape and certain chemical properties. The combination of shape and chemical properties determines which molecules may bind to the binding site. Note that the binding of a ligand to a receptor is independent of sequence identity between the two, but depends on matching geometrical surface and fitting chemical properties. The same binding site may be found in proteins that are structurally very different. This indicates a need for a classification of binding sites that is independent of the families of the proteins in which they are found.

Several databases have emerged in an attempt to create a classification of the protein binding regions that are found across for the protein families. One example is the Structural Characterization Of Water, Ligands and Proteins (SCOWLP) [Tey08] database, which has been developed in the group and which uses information from the SCOP database. SCOWLP contains information about binding sites that has been extracted from the cases of PDB files that contain complexes consisting of a ligand bound to a receptor. These PDB files allow the extraction of geometric information about the binding site.

It has been determined that only a small fraction of residues found in a binding site are necessary for the binding to take place. These residues represent so called hotspots and it is these that best identify a binding site. The protein may bind similar molecules if the same hotspots can be found as in a different protein. Geometrical and chemical properties of other residues may vary, but as long as the hotspots are conserved binding sites may be said to be similar and able to bind the same ligands.

1.2 Motivation

Experimental determination of protein interaction is difficult and time consuming. However, because we know that structurally different proteins from different families may contain similar binding sites, we may be able to determine new interactions between proteins if we are able to determine that two proteins contain the same or a very similar binding site. This would indicate that the two proteins would be able to bind the same ligand, information that could be helpful in inferring new protein functionally, aid in recognition of function and in drug design.

The same binding site may be found in proteins belonging to separate families that are structurally very different. This means that we can not depend on sequence similarity or secondary structure similarity between the proteins if we want to find the same binding site in different proteins. The problem is then to identify a similar binding site by looking for geometric similarity in different protein structures. Given a good geometric fit, conforming physical and chemical properties between superimposed atom pairs should be considered to get a stronger indication of having identified the same binding site.

There are some main challenges presented by the problem of superimposing two binding regions. First of all we are interested in removing outliers such that only the relevant parts of binding sites are aligned. However, it is not clear which atoms that are part of the core, so an initial geometric approach to outlier removal has been taken as a starting point.

A related problem is achieving biologically significant results. These may differ from the best obtainable geometric superimpositions and in such cases a means allowing the method to arrive at something that is biologically relevant must be found.

1.3 Existing approaches

Other methods for solving a very similar problem exist. The most widely used program is MultiBind [SPSNW08]. MultiBind is a novel computational method, for recognition of binding patterns common to a set of protein structures. It performs a multiple alignment between protein binding sites in the absence of overall sequence, fold or binding partner similarity. MultiBind recognizes common spatial arrangements of physico-chemical properties in the binding sites.

The MultiBind method applies an efficient Geometric Hashing technique [WR97] to detect a potential set of multiple alignments of the given binding sites. To overcome the exponential number of possible multiple combinations it applies a very efficient filtering procedure which is heavily based on the selected scoring function.

2 Methodology

The goal of this work is to develop a methodology for superimposing binding site descriptions and to derive a measure that determines how similar they are.

This would allow superimpositions to be made between all the binding site descriptions found in SCOWLP. From these superimpositions and the measure of how similar binding sites are, binding sites could be classified.

The program to perform the superimpositions should be efficient and highly modular to allow changing methodologies and parts of the approach. It should also be simple to parameterize the program so that different approaches may be chosen easily.

Because of the variations found in the same binding site when found in different families, we are interested in the most well suited *local alignment*, not necessarily a global one. A global alignment is an alignment of all atoms from atom set D onto atom set M. With a local alignment, we may discard non-fitting *outlier* atoms from D and M if this allows us to arrive at a particularly good superimposition. Of course the discarding of atoms can only be done up to some limit, because discarding all atoms except one from each set of points, will guarantee us that we have a perfect superimposition. Different approaches to the removal of outliers have been made. Note that in the following the word points will often be used when referring to the coordinates of an atom.

A related but different requirement to the program is that it should be able to find the best fit of one set of atoms onto another when the atoms do not match exactly. In other words, we are not necessarily interested in an accurate fit.

The program should read and write binding site descriptions and superimpositions made in a format similar to the PDB [rcs09]. This would allow visual inspection of data used by the program, valuable for debugging and analyzing input data and resulting superimpositions.

2.1 Data from SCOWLP

Data describing binding sites to superimpose will be extracted from the SCOWLP database. The data includes information about the geometric arrangement of atoms in the binding site, the atom element and also physical and chemical properties.

The description also includes the center of mass of the protein, which will be used together with the center of mass of the binding region to create a vector used to indicate the directionality of the binding region. The directionality vector may be used for aligning the proteins to an initial starting position for starting the superimposition, and to impose user specified constraints on the maximally allowed rotation of one binding region relative to the other. This effectively reduces the search space and prevents some illegal superimpositions from happening.

For a fit to be good the chemical properties of the atoms in the fitted binding site must conform to some constraints, causing some superimpositions to be better than others, and even rejecting some alignments as impossible. A simple approach was taken to initially differentiate two kinds of physicochemical properties: hydrophilic and hydrophobic atoms. A hydrophilic atom will never be able to pair with a hydrophobic atom in a superimposition.

Figure 2 gives a graphical example of the data from SCOWLP.

To limit the number of atoms making up a description of a binding site SCOWLP employs methods for creating *pseudo atoms* from each of the residues found in the binding site. A pseudo atom is a combination of several atoms representing an average of their common location and chemical properties. Specific rules on how to make pseudo atoms from residues are used to reduce the number of atoms in the binding site to a significantly lower

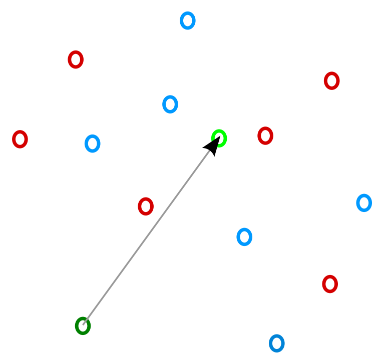


Figure 2: An example of a binding site representation extracted from SCOWLP. Hydrophilic pseudo atoms are shown in blue, hydrophobic pseudo atoms in red. The dark green atom represents the center of mass of the whole protein. From this and the light green center of mass of the binding site a directional vector is created. This vector may be used as a rough indication of where the binding occurs and to limit the size of the possible solution space by putting a limit on the angle between the directional vectors of two binding sites.

number of pseudo atoms. In the process it is noted how many atoms go into the creation of a pseudo atom. Also the number of interactions known for atoms are summed and stored along with each pseudo atom.

Appendix C contains a description of the file format of binding site description files output from SCOWLP.

2.2 Initial approach

Before going into details of the methodology developed, I will briefly outline some of the steps that were taken towards the final methodology.

The two central algorithms used by the developed methodology are variants of *Particle Swarm Optimization* (PSO) [EK95] and *Iterative Closest Point* (ICP) [BM92]. PSO is a heuristic method for optimization of non-linear functions. The development of the technique was inspired by how schools of fish or flocks of birds move synchronously together. The technique is explained in section 3.3. ICP is a method that is used to match two sets of points to each other. It iteratively estimates the optimal alignment of the two point sets and may be used real–time. Section 3.4 explains our use of ICP in arriving at a superimposition.

Efficient removal of outliers was initially a central focus point and also a partial motivation of the choice to use a variant of ICP as the basic algorithm to superimpose two point sets, or as in our case, two sets of atoms. Especially the frICP [PLT06] variant of the algorithm seemed interesting because of the technique for outlier removal described in the paper. The terms *Model* and *Data* used in this paper stuck and are used throughout this report; The

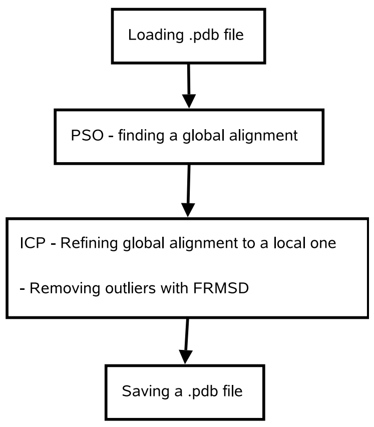

Loading .pdb file
PSO - finding a global alignment
ICP - Refining global alignment to a local one
- Removing outliers with FRMSD
Saving a .pdb file

Figure 3: Initial approach to solving the problem. PSO was used to arrive at a good global alignment where after ICP was used to refine this removing outliers using the FRMSD algorithm. This approach did not prove to function well, though.

term Model refers to the still standing point set and the term Data is denotes the point set that is moved in an approximated transformation bringing it closer to a superimposition onto the Model. Details of the ICP algorithm are described in section 3.4.

As mentioned in [PLT06], the performance and outcome of ICP is dependant on the initial starting positions of the two point sets relative to each other. If the two point sets are initially close to each other the ICP algorithm converges faster than if the point sets start out very differently.

But how to find a good starting point for the alignment? The answer to this came quickly as there had been previous experience in the group with using the PSO algorithm in a program for docking. The approach was to use the PSO algorithm to arrive at a good global alignment of the two sets of atoms and then to refine this by using ICP and removing outliers in the process as depicted in figure 3. Each particle of the PSO was used to create a translation and a rotation of the Data atoms and the *Root Mean Square Deviation* (RMSD) between the Model and Data was calculated and used to indicate the fitness of the particle. In bioinformatics RMSD is a frequently used measure of the average distance between structures.

The PSO algorithm is described in section 3.3.

The approach was tried and applied to an artificially created test case consisting of a rather dense mass of 32 atoms in the file `mix.pdb`. Atoms in this file were copied, translated and rotated and placed in the file `mix2.pdb` and an attempt to align the two was made. This seemed to be a good initial test as the correct result would be a perfect superimposition of one onto the other. Also, it should be rather simple since a perfect alignment of all atoms should involve only rotation and translation. Figure 4 shows the `mix.pdb` example.

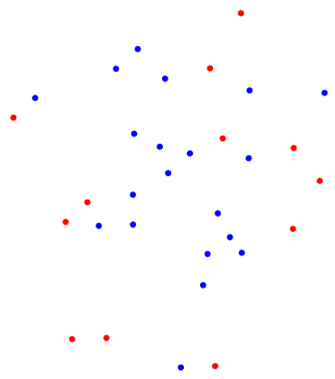


Figure 4: A picture of the artificially created set of atoms I have called `mix`. A copy of this was created, rotated, translated to create a perfectly matching case. The blue dots mark hydrophilic atoms and the red dots mark hydrophobic atoms.

Unfortunately the program did not arrive at a good superimposition of the mix test case, often discarding many of the atoms in an alignment and taking a long time to arrive at a result. This led to the conclusion that a good global alignment is not necessarily a good start for the ICP algorithm and a new approach was sought.

2.3 Final approach

The idea of the final method was to use each particle of the particle swarm to create a transformation of the Data atoms and then to use ICP to refine the alignment of this constellation as before, but for each particle. When the ICP algorithm converges a fitness value is returned which indicates how well the remaining atom pairs are superimposed to each other.

This new approach functioned much better. The mix test case described above was aligned in only a few seconds, including all of the 32 atoms. The RMSD measure returned by the ICP seemed to guide the particle swarm through the solution space quite well.

3 Central algorithms

In this section I will describe each of the parts of the methodology that has been developed in detail. An overview of the whole process from loading two binding site descriptions to saving a superimposed output can be seen in figure 5.

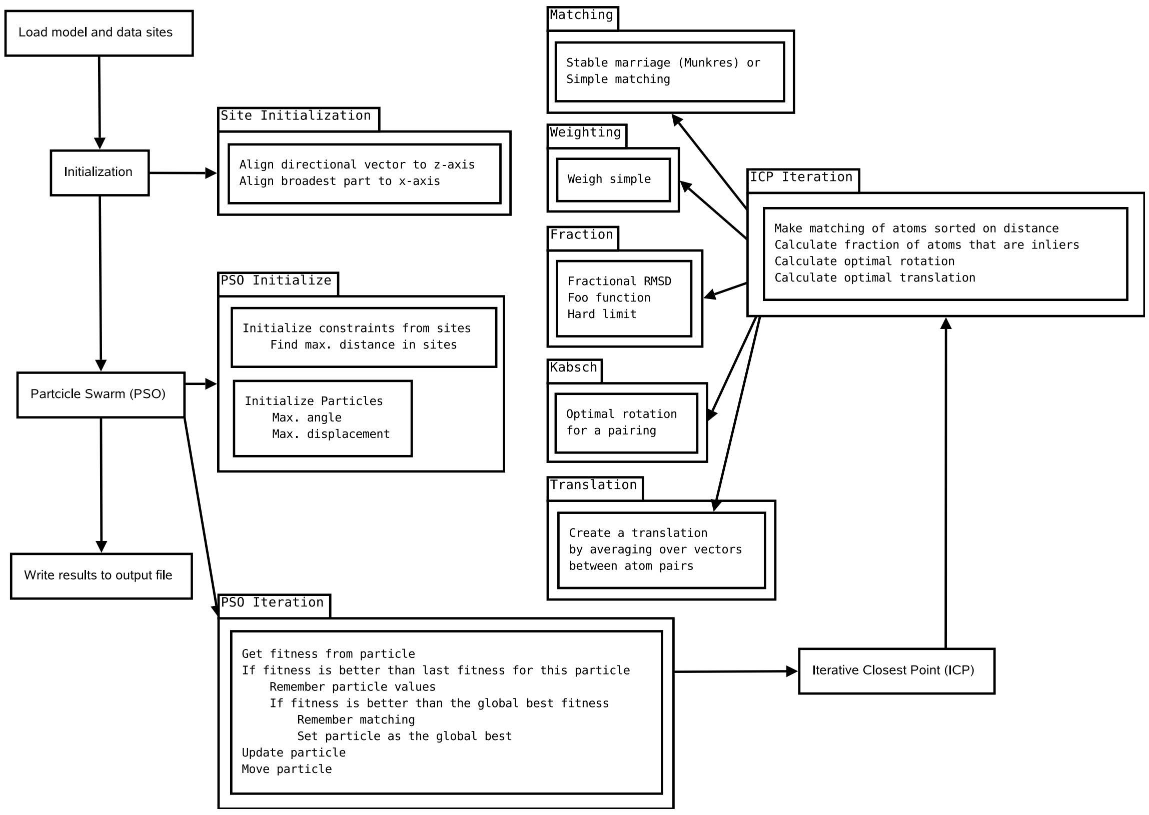

Load model and data sites
Initialization
Partcicle Swarm (PSO)
Write results to output file
Site Initialization
Align directional vector to z-axis
Align broadest part to x-axis
PSO Initialize
Initialize constraints from sites
Find max. distance in sites
Initialize Particles
Max. angle
Max. displacement
PSO Iteration
Get fitness from particle
If fitness is better than last fitness for this particle
Remember particle values
If fitness is better than the global best fitness
Remember matching
Set particle as the global best
Update particle
Move particle
Iterative Closest Point (ICP)
ICP Iteration
Make matching of atoms sorted on distance
Calculate fraction of atoms that are inliers
Calculate optimal rotation
Calculate optimal translation
Matching
Stable marriage (Munkres) or
Simple matching
Weighting
Weigh simple
Fraction
Fractional RMSD
Foo function
Hard limit
Kabsch
Optimal rotation
for a pairing
Translation
Create a translation
by averaging over vectors
between atom pairs

Figure 5: Overview of the main steps performed when making a superimposition of a Data binding site to a Model. Starting in the top left corner, the files describing the Data and Model binding sites are loaded and initialized to a starting position. Now the iteration of the Particle Swarm begins. For each particle of the Particle Swarm a conformation of the Data binding site is created and used as input for the ICP algorithm which consists of creating a matching of pairs between the two molecules, removing outlier pairs and finding and optimal transformation of the Data binding site that will bring it closer to the Model. The ICP algorithm involves creating a matching of atom pairs, weighting how well atoms fit, determining the outlier fraction and finally the creation of an optimal rotation and translation to superimpose the Data to the Model. The ICP algorithm will converge and return a fitness value that is used to set the particle fitness in the Particle Swarm. This repeats for each particle.

For several of the steps in figure 5 there are program options that may be used to specify different methods of performing the step. Appendix B provides a detailed description of the program options that can be used to select variations or change parameter values.

Overview Before going into the details of each step I will briefly describe the main steps of the program as depicted in figure 5.

After loading the files containing the binding site descriptions, the Model and Data sites are initialized. This includes moving the sites to the origin of the coordinate system and possibly also aligning the directional vectors of the sites. Also, the set of constraints to apply to a possible superimposition is initialized.

Next, the iteration of the PSO commences. During each iteration, each particle is used to initialize the Data binding site to starting point for ICP. With this starting point, the ICP algorithm is run for a number of iterations until it converges to an alignment of the atoms. The ICP iterations include methods for creating point pairs, the weighting of how well point pairs fit, removal of outliers and calculation of the optimal transformations to align the point sets. When the ICP algorithm converges, a fitness value is returned (the RMSD, for example) and it is this value that will be used to determine the fitness of the current particle. In this way the particle swarm is directed by the fitness values of the superimpositions created by the ICP algorithm.

Finally the superimposition of the Model and Data binding sites are written to a file. This resulting superimposition can be viewed in a viewer capable of visualizing files in the PDB file format.

In the following sections a more thorough description each of the steps presented in figure 5 is made.

3.1 Loading of Model and Data

The first step is to load the Model and Data binding sites from data files extracted from SCOWLP. The loader creates a representation of the binding site similar to that depicted in figure 10 from files in a format similar to the PDB file format [rcs09].

3.2 Initialization

The first step in the initialization is to translate the atoms of the Model and Data binding sites such that the protein centroid is placed in the origin of the coordinate system.

Next, the directional vectors of both sites are aligned with the `z-axis` and finally the broadest part of each binding site is found and the atoms are rotated so that the broadest part of the binding site lies along the `x-axis`.

For the Data binding site, this initial positioning is used as a starting point for each of the transformations that will be derived from each particle. The Model binding site will remain fixed in this initial alignment.

Finally, constraints limiting the allowed length of translation of the Data points are created from the coordinates of the binding sites. These constraints allow exclusion of superimpositions which cannot be well fitting.

3.3 Particle Swarm Optimization

The basic idea of PSO [EK95] is to represent the parameters of the function to optimize as particles that fly around in n dimensional space, where n denotes the number of parameters to the function to optimize. Each particle has a position, a velocity and a recollection of the previous best position of the particle (*pBest*). A fitness function determines the fitness of a particle by using the particle parameters to calculate the function to optimize and return a value denoting how good the function result is. The particle that has parameters that give the best result of the fitness function is set to be the globally best particle (*gBest*).

Before iteration begins, the particles of the swarm are initialized with random values such that they are uniformly distributed in the solution space. The pBest position of each particle is also initialized with a random value.

Each time the particles in the swarm are moved, the *pBest* and *gBest* particles are used to update the velocity of each particle and hereafter the particle is moved by adding the velocity to the current particle position.

There are a number of variations of the basic PSO algorithm, some of which are described in [CD01]. I have chosen to implement a simple variant of PSO inspired by [HE02]. The implementation uses just one swarm and the global best particle is set after the evaluation of each particle, instead of waiting until all particles have been evaluated in an iteration.

The formula used to update particle velocity is the same as used in the original paper on PSO [EK95]:

$$V = V + (C * rand() * (pBest - P)) + (S * rand() * (gBest - P)) \quad (1)$$

Where V and P are the particle velocity and position, C and S are user specified values used to tune the cognitive and social components of the particle velocities, $rand()$ is a function returning a random value in the range $[0 - 1]$ and $gBest$ and $pBest$ are as described above.

Intuitively, the cognitive and social components can be used to tune how fast the particles should converge towards a global optimum, i.e. the *gBest* particle. A larger social component would cause the particles to move toward the same point fast whereas a larger cognitive component would mean that the particle swarm searches a larger volume of the solution space and therefore convergence is slower. Depending on the problem at hand fast convergence may be desirable or not.

```
Initialize particles according to constraints
for each particle
    if particle fitness is better than previously for this particle
        set best particle fitness
        if fitness is better than the global fitness
            set particle to new global best
    update particle
    move particle
```

Figure 6: The basic steps of the PSO algorithm

The basic steps taken in my implementation can be seen in figure 6. Each particle in my implementation of the PSO is initialized with 7 parameters: 3 describing a translation vector, 3 that describe a vector to rotate about and 1 specifying the angle of rotation. These parameters are used to transform the initial configuration of the Data point set to a starting point for the ICP algorithm. When the ICP algorithm converges, a fitness value is returned. This value is then used to set the *pBest* and *gBest* particles.

It may happen that the running of the ICP algorithm results in an alignment of the Data set to the Model that does not satisfy the constraints. In this case the particle is restarted from its previous best position. A parameter (`--wiggle`) exists to allow the user to specify a small random displacement of this restart position.

3.4 Iterative Closest Point

Iterative Closest Point [BM92] is an algorithm for iteratively aligning two sets of points. ICP is widely used for 2- and 3-dimensional alignment of point sets when an initial estimate of a good starting point is known. ICP is computationally efficient and may be used in real-time applications.

The basic steps of the algorithm are as follows:

1. Create a pairing between points between the two point sets
2. Estimate a transformation that will align the point sets
3. Apply the rotation and the translation to the two point sets
4. iterate

ICP was chosen as a means to refine a superimposition mainly because there are efficient variants of the ICP algorithm that deal well with the issue of removing outliers [CSK05] [PLT06].

3.4.1 Creating the pairing

To find and remove outliers from the pairing, I initially implemented a variant of the *Fractional Iterative Closest Point* (FICP) [PLT06] algorithm. It turned out that the removal of outliers was too aggressive using this approach, but a better approach was found and the framework was changed so that it is easy to experiment with other functions for removing outliers. See section 3.5 for details.

When calculating the optimal rotation and translation to apply to the Data point set, only the points regarded as inliers are used for the calculation, but the transformations are applied to all points of the data set. This allows for new pairings to be created as the Data point set moves closer to the Model.

3.4.2 Estimating optimal transformations

To estimate the optimal rotation between the two point sets, I have used an implementation [Ho09] of the Kabsch algorithm [Kab76].

Given a pairing of points in space, the Kabsch algorithm will calculate a rotation matrix that, when applied to the Data point set, will rotate it so that the RMSD between the point sets is minimal. Note that the computed rotation minimizes the RMSD distance between point pairs created and not between the two point sets in general. This means that the atom pairing used is of high significance for the outcome.

The calculation of the optimal translation to apply to the now rotated point set is done by calculating an average translation vector between the paired points using the following formula:

$$T = \frac{\sum_{i=0}^{n}(M_i - D_i)}{n} \tag{2}$$

Where n is the number of pairs that have been matched and M_i and D_i are the corresponding paired points from the model and data point sets. Applying this vector to translate the Data point set will result in a further minimization of the RMS distance between the Model and Data.

The iteration of the ICP algorithm stops when the maximum number of iterations has been reached or when the resulting rotation is the identity matrix and the translation is $(0, 0, 0)$.

3.5 Pairing of atoms

There are several different methods one can employ when creating a pairing between the two sets of atoms. Some examples are ray-casting and creating surface normals to indicate in which direction a given point of one set is likely to find its match. The approach initially taken by the Superimpose program developed in this work, was to find the nearest neighbour of the other point set. This approach was chosen because it is fairly general and because the nature of the data describing the binding region is not always dependable.

To restrict a pairing of a hydrophilic and a hydrophobic atom, the distance calculation is artificially modified to be very large between incompatible atoms, assuring that pairings between these atom types will not be included.

3.5.1 The KD-tree

The initial approach to perform a pairing between the model and data point sets was to use a KD-tree [Ben90]. A KD-tree is a data structure that represents a set of n points in K-dimensional space, in this case in 3-dimensional space. It can be used for efficiently finding the closest point to a given point. Lookup of a closest point takes $O(\log n)$ time and the creation of the tree takes $O(n \log n)$ time.

The motivation for using a KD-tree was its efficiency. Because transformations are applied only to the data point set, the KD-tree can be calculated just once for a further improvement of efficiency.

There were some problems with using a KD-tree: It is often the case that several points will have the same point as their closest point. This means that the translations calculated will arrive at a good average superimposition distance between points, but rarely an exact match. Therefore one-to-one point paring was tried next.

3.5.2 Using a grid

A simple way to avoid the problem of pairing several points with the same point is to calculate a grid of all distances between atoms and choosing point pairs by finding the smallest distances in the grid. If several atoms have the same atom as their closest atom, then only the atom with the smallest distance is regarded as a pair. All other atoms are discarded.

As it turns out this approach also has some problems. First, it is rather time consuming, taking $O(|M| * |D|)$ time for each time the pairing is performed. Second, there is a risk of discarding too many atom pairs when many atoms have the same atom as their nearest neighbour. If we are very unlucky all atom pairs but one will be discarded. The result is that the transformations computed to superimpose the Data set to the Model become skewed.

The computation of the grid led to the next approach.

3.5.3 The Munkres algorithm

The Munkres algorithm [Mun57] also known as the Hungarian method solves the *Assignment problem*: Given N workers and T tasks find the optimal assignment of tasks to workers so as to minimize the total time taken to perform all the tasks. If the number of workers and tasks are the same the problem is denoted the *Linear Assignment problem*.

The Munkres algorithm was applied to the grid of distances between the atoms of the two binding regions, to arrive at a pairing such that each atom is paired uniquely to exactly one other atom and the sum of the distances between atoms becomes minimized.

This turned out to be a good approach.

Unfortunately the time complexity of the Munkres algorithm is rather high taking $O(N^4)$ in its original form. This has been improved to $O(N^3)$. The implementation used in Superimpose uses the improved version [Wea09].

A like approach to creating unique pairs from a grid of all distances between atoms of the two binding regions, is to use the Gale-Shapley [GS62] stable matching algorithm. This has not been tried, but would be interesting, since the Gale-Shapley algorithm may be faster. However, some tricks would have to be used since the algorithm expects to match elements from two sets of the same size, and it is seldom the case that there are the same amount of atoms in two different binding regions.

3.6 Determining outliers

The class `matching` which implements the matching of points using one of the methods described above, makes available a list of atom pairs sorted in ascending order on the distance between the pairs. This distance may be modified by the weighting of how well the two atoms fit together. The list is used by the methods of determining inliers described in the following subsections, by setting an index which separates inliers and outliers.

3.6.1 Fractional RMSD

The calculation of the Fractional RMSD value was taken from [PLT06]. The idea is to minimize the following expression:

$$\frac{1}{f^{\lambda}}\sqrt{\frac{1}{|D_f|}\sum_{p\in D_f}||T(p)-u(p)||^2} \tag{3}$$

where $f \in [0,1]$ is the fraction of inliers, D_f are the closest $f|D|$ point pairs, $T(p)$ is the transformation applied to each point in the Data point set, and $u(p)$ is the point from the Model with which p is paired.

The term $\frac{1}{f^{\lambda}}$ is used to balance the RMSD term. λ is an empirical value that can be changed by the user. The authors of [PLT06] recommend setting $\lambda = 0.95$ when matching points in three dimensions.

Preliminary experiments using this method showed that the removal of outliers seemed to be too aggressive using $\lambda = 0.95$. Setting the lambda value significantly higher increased the amount of inliers. This may be due to the rather small point sets that we used compared to the point sets used in [PLT06].

The initial experiments with fractional FRMSD inspired the search for a different function for outlier removal, mainly because FRMSD was very sensitive to changing lambda values.

3.6.2 The QScore

The function dubbed the *QScore* was found in [KEHK04]. The computation is a maximization of the following formula:

$$\frac{P^2}{1 + (RMSD/R_0) * |M| * |D|} \tag{4}$$

Where P is the number of pairs used for the calculation of the RMSD value, $|M|$ and $|D|$ are the cardinalities of the Model and Data point sets and R_0 is an empirical value that is specified by the user serving to balance the significance of the RMSD value and the number of aligned pairs P.

The strength of this function for the calculation of inliers and outliers is, that it takes into account not just the number of atom pairs and the RMSD value, but also the number of points in the Model and Data point sets.

By changing the value of the R_0 parameter it is possible to weight the significance of the RMSD score in the value of the measure. The larger R_0 is the less significant the RMSD will be.

The following two subsections describe two simple approaches to determining which pairs should be regarded as outliers.

3.6.3 Limit

This method is a very simple way of specifying the border between inliers and outliers in the list of sorted atom pairings. The user specifies an Ångstøm distance and all atom pairs with a distance between them that is less than this value will be regarded as inliers.

This method is highly dependant on the initial starting position of the two sets of atoms. It generally does not function well.

3.6.4 Fixed

The fixed pairing was introduced to allow for refinement of manually created superimpositions of binding sites. A file containing the expected pairings specifies the only pairing that will be considered. This pairing is used in refining the alignment with ICP.

3.7 Calculating a fitness value

Initially the *Root Mean Square Distance* (RMSD) was used to calculate a fitness value indicating the "closeness" of the two atom sets. This is a commonly used measure for quantifying the distance between two geometrical shapes. It is

$$RMSD(P,Q) = \sqrt{\frac{\sum_{i=1}^{n}(P_i - Q_i)^2}{n}} \tag{5}$$

Where P_i and Q_i denote the Cartesian coordinates of matching atom pairs.

Later, a similar measure was found that also includes information about the difference in internal distances between the two sets of atoms. This was termed the DRMS after the paper in which it was initially found [SL03]. The formula is:

$$DRMS(P,Q) = \sqrt{\frac{2}{n(n-1)} \sum_{i=2}^{n} \sum_{j=1}^{i-1} (d_{ij}^{P} - d_{ij}^{Q})^2} \tag{6}$$

This measure compares the matrix of distances between all points within each structure. For a point set P this matrix is defined as the distances between all points:

$$d_{ij}^{P} = ||p_i - p_j||_2 \tag{7}$$

The cost of computing this measure grows quadratically as the size of the proteins for which it is calculated increases.

3.8 Weighting

A simple form of weighting was been implemented to try and take into account additional information known about a binding site. Time constraints did not allow thorough examination of how to apply weights and further work should be done here.

For each atom in the binding site, the number of interactions in a known binding is recorded in the data received from SCOWLP (see appendix C).

This information is used to calculate a similarity measure between this atom and candidate atoms for a pairing.

The reasoning behind this approach is that atoms will typically pair well with atoms that have almost the same number of interactions. For weighting the similarity, the following formula has been used:

$$\frac{|I_D - I_M|}{I_D + I_M} \tag{8}$$

Where I_D and I_M denote the number of interactions between the Data and Model, respectively.

This weighting is used to modify a user specified fraction of the calculated distance between the two atoms, such that the distance is decreased if two atoms match particularly well. Figure 7 shows a graphical representation of how these weightings are used to modify the RMSD distance.

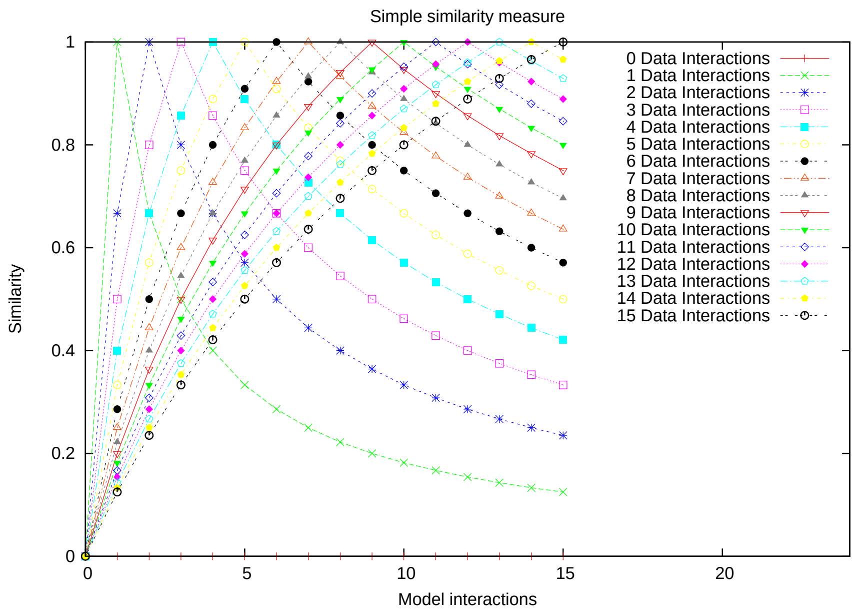


Figure 7: Similarity measures using known interactions of the binding sites. When a model atom pairs with a Data atom that has the same number of interactions the whole user specified fraction of the distance between the two atoms can be removed. This is shown in the graph as a similarity of 1.0

4 Some implementation details

Superimpose has been implemented as a command-line program using the C++ language. C++ was chosen as implementation language because an efficient implementation was wanted. It should be able to perform comparisons between all binding site descriptions found in SCOWLP within a reasonable amount of time.

The excellent `boost::program_options` library was chosen as a means to configure the run of Superimpose. It allows options to be specified in a configuration file (defaults to `$HOME/.superimposerc`) or as command line options. Command line options will override options set in the configuration file. This way of configuring the program is flexible and allows for easy parameterization of the program.

User specified options to the program are collected in the `config` object which is implemented as a singleton. It is used mainly by the `factory` class which creates objects from classes according to the settings found in the `config` object.

The GNU Scientific Library [gsl03] has been used for creating high quality random numbers.

Polymorphism using virtual methods has been used liberally throughout the program to implement the different methodologies. It has been argued that the lookup that C++ makes into a vtable for each call to a virtual function causes a large overhead, but this has not been found to impact the performance significantly.

I chose `valarrays` which are part of the STL to hold the values of each particles position, best position and direction. These are well suited for this purpose as they have been designed for efficient numerical computations and also define convenient operators for arithmetic operations on all members.

Finally, I would like to mention the design decision made to simplify memory management. From early on I decided to create as many as possible of the objects needed by the program before beginning the superimposition. The motivation was to simplify memory management by avoiding memory allocations and deallocations being scattered across the program in many different files.

The `std::auto_ptr` template class is used to hold all of the central objects that are allocated beforehand. This guarantees that objects will be deallocated when the scope is left. The allocation of the objects is done in the `main` function.

Please refer to the doxygen generated documentation in the `doc/` directory for further details.

4.1 Memory leaks and profiling

To verify that there are no memory leaks the program has been run with various parameter settings using the `--leak-check=full` option of the `valgrind` program [NS07]. This helped to find and correct a few mistakes.

To direct possible future optimizations a profiling of the program was made using `gprof` [GKM04]. The results clearly indicate that the pairing of atoms is the most time consuming, both when using the `munkres` and the `simple` methods of pairing. It may be possible to achieve increased performance by using the Gale-Shapley stable matching algorithm instead of the Munkres algorithm to create unique atom pairs.

A considerable amount of execution time is spent on applying weights to the atom pair distances. This can be optimized because weights are calculated anew for each time a new distance is calculated. Instead a grid containing weightings of all possible atom combinations should be pre-calculated, and the values in the grid should then be applied to newly calculated distances.

5 Evaluation

The evaluation of the program has been done in two separate parts. In the first part the program was compared with two cases of superimposition that were manually created. The goal of these initial tests was to assert that the program works correctly, and to evaluate its performance on two known cases where the expected outcome is known. The superimpositions should be relatively easy as the binding sites used are from the same family.

The second approach was of a more high throughput nature and involved an extraction of 270 cases of known binding site interactions from SCOWLP. By running the Superimpose on this data, a representative parameterization on this dataset was sought. Also, a comparison with the MultiBind program was made.

The superimpositions made in this part should be significantly more difficult because all binding site comparisons are between binding sites from different families.

5.1 Two specific cases

Several methods of creating atom pairs and of separating inliers and outliers have been tried. The following methods proved to be superior to other approaches for arriving at superimpositions, and therefore the tests run will use these methods:

- Using the *munkres* method of creating point pairs.
- Using either the *QScore* function or the *FRMSD* function for defining how to separate inliers and outliers.

A goal of these first tests was to see how the simple weighting methodology influenced the performance on the two cases.

For these tests the Phosphotyrosine-binding domain (PTB) of the PDB files with codes `1x11`, `1aqc` and `1shc` were used. The binding sites of the PTB domains were chosen because these had been studied well before in the group, and because the domains and therefore also the binding sites are relatively small, so that the program would be able to arrive at a result relatively fast. Also, the proteins in whcih the domains are found belong to the same family, meaning that there should be some structural similarity between the binding sites.

The quality of superimpositions achieved has been evaluated by looking at the closest pseudo atom pairs that the program finds and comparing these with a list of expected atom pairs. This method was chosen in favour of a calculation of how close the two structures are to each other because the main focus of Superimpose is on finding the most biologically significant superimposition, not the best geometrical one. If the program returns a superimposition that uses a large number of the same pairs as the manually created superimposition this would be a good indication that we are able to arrive at a superimposition that is comparable to the expected one.

5.1.1 Creating the expected pairs

Manual superimpositions were created of the PTB domains of `1x11` vs. `1aqc` and `1x11` vs. `1shc` using Accelrys Discovery Studio Visualizer [SI09].

The serial numbers of the best atom pairs of these two superimpositions were noted into a file. Using this file to specify a fixed pairing, Superimpose was run on the same input data to arrive at the best superimposition using these pairs. This was done by specifying the `--icp_fixed_pairing` option to Superimpose. The resulting number of atom pairs from these refinements are shown in table 1.

The pairings written to this file were then compared with the pairs that the Superimpose program arrived at when running with different parameters.

Comparisons	Pairs	RMSD (Å)
`1x11` vs. `1shc`	14	2.4265
`1aqc` vs. `1x11`	20	3.0994

Table 1: The number of pairs and the resulting RMSD value achieved by refinement of superimpositions created manually by running Superimpose with the `-icp_fixed_pairing` option. The pairs were noted into a file and used for comparison with superimpositions created by Superimpose

The parameterizations of the program that are common for these tests are as shown in figure 8. For a description of the meaning of these parameters, please see appendix B.

```
initialize          =           align

#-------PSO--------
pso_particles       =              20
pso_iterations      =              20
pso_cognitive       =            2.10
pso_social          =            2.15
pso_min_fitness     =            0.01
pso_wiggle          =             0.3
pso_divisor         =              42

#-------ICP--------
icp_iterations      =              64
icp_min_points      =               5
icp_matching        =         munkres
icp_max_angle       =              60
```

Figure 8: The parameters of the program that are common for all the tests run with the program

The configuration settings that were varied when running the Superimpose program in these tests were the application of weights and the method of removing outliers. Four different combinations of configuration settings were tried: Outlier removal using both the QScore and the FRMSD methods, both with and without applying weights.

When using FRMSD the λ value was set to 7. Using the QScore function the R_0 value was set to 3. When using weights, the amount of the RMSD that can be influenced by the weighting function was set to 0.9. These values were chosen, because initial tests on different cases showed the best results using these values.

For each of the two test cases 40 runs of the program were made using the four different configurations. The worst, best and average number of expected pairs were recorded along with their corresponding RMSD values. The only parameter that changes for each run is the seed for the random

1aqc vs. 1x11	Worst		Average		Best	
	Pairs	RMSD (Å)	Pairs	RMSD (Å)	Pairs	RMSD (Å)
QScore, no weights	19/19	0.841	19/19	0.841	19/19	0.841
QScore, weights	19/18	0.198	19/18	0.197	19/19	0.186
FRMSD, no weights	17/17	0.412	17/17	0.412	17/17	0.412
FRMSD, weights	19/19	0.198	19/19	0.197	19/19	0.190

1x11 vs. 1shc	Worst		Average		Best	
	Pairs	RMSD (Å)	Pairs	RMSD (Å)	Pairs	RMSD (Å)
QScore, no weights	15/0	3.212	17/8	3.02	18/11	2.087
QScore, weights	20/4	1.079	20/5	0.980	20/6	0.972
FRMSD, no weights	24/2	5.196	24/4	5.195	24/5	5.159
FRMSD, weights	22/5	1.061	22/5	1.018	22/6	0.927

Table 2: Results of runs of the Superimpose program using on the PTB binding sites of `1aqc` vs. `1x11` (top) and the PTB binding sites of `1x11` vs. `1shc` (bottom). There are two numbers given in the Pairs columns. The first number indicates the number of pairs found that are regarded as inliers. The second number indicates how many of these pairs that are also expected pairs.

number generator.

The results of these runs can be seen in table 2. The Pairs column of these two tables contain two numbers where the first indicates the number of pairs found that are regarded as inliers and the second number indicates the number of these pairs that are also found in the expected pairing. Note that the RMSD value in the case of weighted runs is not the actual RMSD value, but rather the RMSD value calculated from distances modified by the weighting.

5.1.2 Evaluation of results

The superimpositions of `1aqc` vs. `1x11` are generally good. In the worst case 17 of the expected 20 pairs are found in the superimposition and in the best case 19 of 20 expected are found.

For the QScore function the application of weights does not improve results significantly. In the case of the FRMSD method of removing outliers results are improved with the application of weights. Using FRMSD with no weighting gives the worst results.

It is notable that the RMSD values of the superimpositions made by Superimpose are much better than the RMSD value of the manual superimposition (3.0994 Ångstrøm).

The superimpositions of `1x11` vs. `1shc` are not so good. The worst case shows that we find none of the expected pairings. In the best case 11 of the expected 14 pairs have been found in addition to 7 not expected pairs.

The QScore method of outlier removal seems to be slightly better than the FRMSD method. In both cases slightly more of the expected pairs are found in the average and best cases using the QScore than when using the FRMSD measure.

The application of weights does not seem to better the results. When using the QScore function the application of weights even worsens the results. This indicates that the method of weighting does not have a positive effect on the results.

It is suspected that the reason that the superimpositions of `1aqc` vs. `1x11` is much better than superimposition of `1x11` vs. `1shc` is that there exists a superimposition of points that fits better geometrically than the manual superimposition in the case of `1x11` vs. `1shc`. This indicates that more strict chemical constraints could help to better the results.

The results of these two tests only gives a weak indication about the actual performance of the program, mainly because there are only two test cases. The initial plan was to look at more cases from the PTB domain, but this decision was not taken in favour of a more high throughput method that would be able to give a better picture of the performance of the program under different settings.

Another reason for choosing this route was that the manual creation of the expected pairs was rather time consuming making it not feasible to create many test cases.

5.2 Testing on a larger dataset

A larger dataset was extracted from SCOWLP and used for two purposes: to see with which parameter settings that Superimpose performs the best and for performing a comparison of Superimpose and MultiBind.

Having a larger dataset the results should be more significant and possible tendencies should be easier to find. Also, there was an interest in the group in a method for curating data from SCOWLP.

The protein structures in the PDB files corresponding to the extracted binding site data from SCOWLP are complexes consisting of a receptor to which a ligand is bound. In the extracted data receptors in two complexes that contain the same binding site are from the same family. This means that a structural alignment of the receptors should be straightforward using a program for structural alignment of proteins. The ligands, however, are from different families suggesting that superimpositions would be more difficult.

When the receptors are aligned the two ligands will be placed closely together in a superimposition that is representative of a superimposition desirable to find using a program like Superimpose or MultiBind. This happens because the receptors are in the same family and therefore are expected to have the binding site located in the same surface area. By aligning the receptors and noting how the ligands are placed on top of each other a reference data set can be created and used to measure the quality of an alignment obtained by running Superimpose or MultiBind. The closer the result of either of these programs is to the reference, the better. This method effectively allows for comparison of binding sites across family borders.

Note that neither Superimpose or MultiBind would have information on a receptor available when trying to align to binding sites in a real setting. This is simply information that we are lucky to have in our test dataset.

As with the simpler test cases it was chosen to create lists of expected atom pairs between the ligands belonging to the two aligned receptors and use these to represent the ideal alignment. If the superimposition is a good fit it should include many of the pairs also found in the list of expected pairings, just as in the previous section.

It should be mentioned that this methodology does have some problems, however. Results are dependant on the distance within which pairs are created and also on how representative the superimposition of the ligands are of an actual binding when the receptors are aligned. Some variation in size of binding sites and in the position of the binding sites is expected, but generally the alignment of the receptors should place the ligands in a fashion that is representative.

Even though there may be some inaccuracies by using this method the goal was to get a fairly good picture of how the two programs compare. Also, we are able to create a smaller representative subset of the initial dataset as defined by the extraction from SCOWLP which can be used for further tests. This smaller set of data could be useful in future experiments.

5.2.1 Creation of the reference dataset

First, 270 known binding site superimpositions were extracted from the SCOWLP database and the PDB files containing the corresponding protein molecules on which the binding sites are found were downloaded from the RCSB Protein Data Bank. A criteria for extracting exactly these was that the binding site size lay between 10 and 40 atoms. Most information on binding sites in SCOWLP is about binding sites in this size range.

The MAMMOTH [LLMO05] program was used to create a alignment of the two receptors as depicted in figure 9. MAMMOTH structurally aligns two proteins using a methodology that involves decomposing the proteins into short peptides. The computation of an alignment is typically fast, allowing us to align the set of PDB files within half an hour.

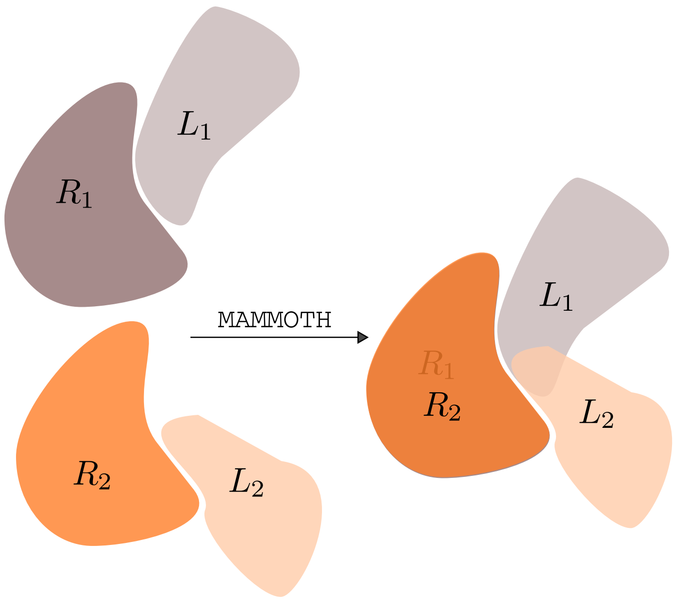


Figure 9: A schematic view of how the backbone of the receptors R_1 and R_2 are aligned using `MAMMOTH`. This results in an alignment of the ligands L_1 and L_2 which is used to create the file containing the expected pairings. This is done by calculating all distances between residue centroids of L_1 to residue centroids of L_2 and then removing pairs with a distance larger than a certain cut-off distance. Only residues of L_1 and L_2 within a certain distance of the receptors are considered for creating the expected pairs. Note that the contact between R_1 and L_1 is slightly larger than that between R_2 and L_2 and also placed slightly differently. This illustrates that the definition of a binding site can vary slightly both in size and location.

The program `create_expected_parings` was run on the data created with MAMMOTH. It creates a list of residue pairs that lie within a certain distance of each other. The program functions in two steps. First, centroids are created from each residue in the two ligands and the two receptors. Then the ligand centroids that lie within a certain distance of the receptor are isolated. These closest ligand centroids are taken to be the atoms that represent the atoms that make up the binding site. The expected pairs are formed between these and written to the expected pairs file. The distance from a receptor within which ligand centroids are selected was set to 8Å. The distance between a residue centroid of L_1 to a residue centroid of L_2 within which pairs are formed was set to 3Å. Figure 10 shows two receptors aligned with MAMMOTH and the corresponding ligands.

This approach is slightly different from the previously described approach using expected pairs in that closest residue centroids are used for creating the expected pairs. This is somewhat crude compared to before

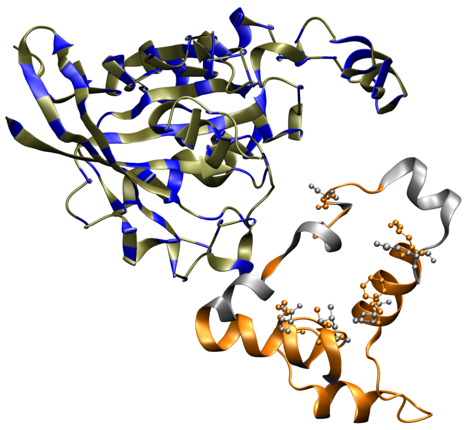


Figure 10: An example of an alignment made by MAMMOTH. The backbones of the receptors are shown in blue and green and as can be seen they align very well. Below in grey and orange the backbones of the corresponding ligands are shown. Some of the residues that have been found to be pairs by looking at their centroids are shown with a ball and stick.

where a single atom reported in the output of Superimpose was used for comparison. The reasons for choosing to rely on residue centroids are several.

First of all MultiBind reports only the corresponding residues of the pseudo atoms that have been paired, along with the distance between the two and the bonding method observed. However, it may be possible to deduce the exact interacting atom by looking at how MultiBind defines pseudo atoms and comparing this with the MultiBind output. This is rather involved and has not been tried.

Another reason is that pseudo atoms are defined differently by SCOWLP and by MultiBind. This means that the results of the two are not directly comparable as the interacting pairs reported by the two programs will differ even if the exact same superimposition was made.

To be able to compare the output with the expected pairs the atoms reported by Superimpose are translated into their corresponding residues before being compared with the expected pairs.

It was decided to use the two ligands for creating atom pairs and not the receptors, because it was thought that the ligands are slightly less sensitive to variations in binding site locations than the receptors.

5.2.2 Collecting test data

MultiBind and Superimpose were run on the test data using different parameterizations.

Figure 11 gives an overview of the whole test process including creation of the expected pairs, running of Superimpose and MultiBind, the extraction of results placing them in a database. All steps in the process are bound together in a rather large python script `test.py` which can execute the various stages of the test data collection via arguments to the `--action` switch.

The final step of the data collection is to place results of the test runs into a `sqlite3` [htt09] database. This database can be conveniently queried using SQL allowing for easy access to different views of the test runs.

Special efforts have been made to assure that the superimpositions that are created by Superimpose and MultiBind may be visually inspected. This is important for veryfying the quality of the superimposition.

5.2.3 Looking at Superimpose

Using the larger dataset it was a goal to establish what is the best fitness measure and method of outlier removal including the best combination of the two. This involved testing 4 different combinations of fitness calculation and outlier removal: `DRMS-FRMSD`, `DRMS-QScore`, `RMSD-FRMSD` and `RMSD-QScore`.

A second but lesser goal was to find out what the best R_0 value for the QScore and the best lambda value for FRMSD are. This was done by running Superimpose with lambda values in the range $1.0 - 5.0$ and R_0 values in the same range.

In the following I will use the term lambda to refer to both the lambda value setting for the FRMSD and the R_0 value setting for the QScore. This is because the option to superimpose to set this value is called `--icp_lambda`. Another option specifies if this value is applied to the FRMSD or the QScore.

Other parameter settings of Superimpose are as for the two simple test cases shown in figure 8.

A number of queries was made into the database to establish which of the 4 mentioned combinations performs best. Unfortunately the results where not clear when looking at results from Superimpose alone, indicating that there may be noise in the data. I decided to look at the performance of each of the 4 combinations in cases where both MultiBind and Superimpose do well. The agreement of the two programs can be seen as a sort of “filter” for removing inaccuracies.

The criteria in the query was to look at how the *good fits* of Superimpose are distributed among the above categories. Here *good fits* are defined as

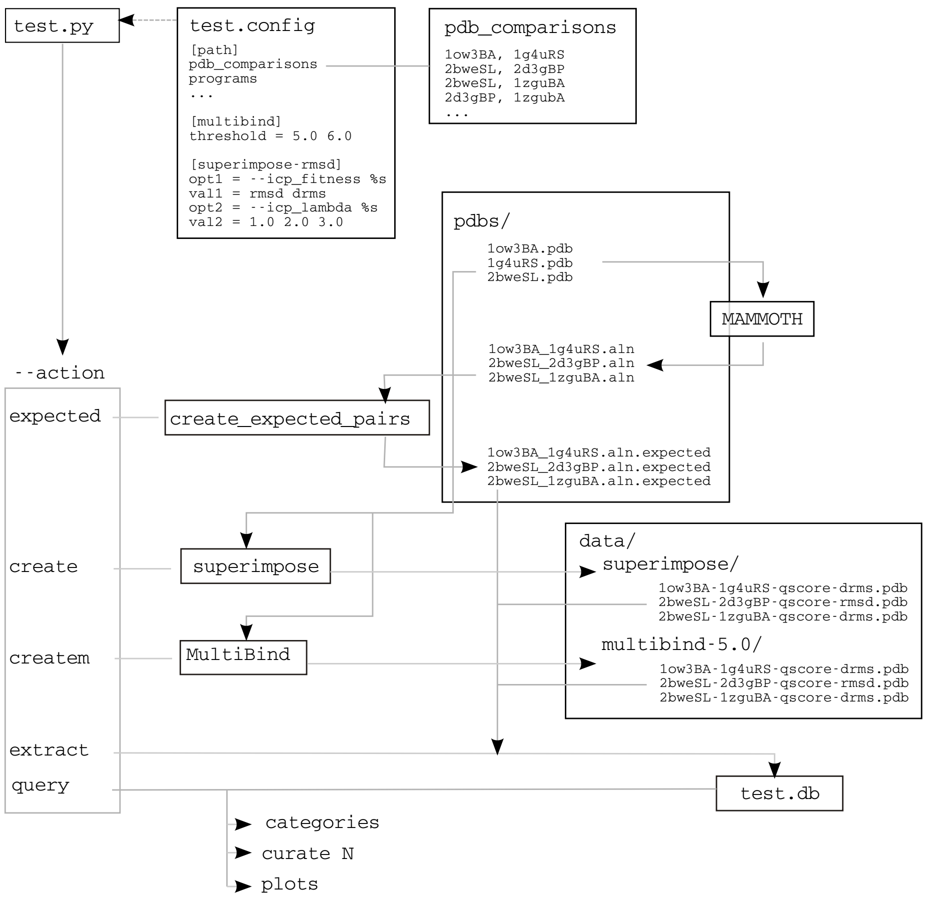


Figure 11: A simplified view of the method used to evaluate the accuracy of the two programs compared. The `test.py` program reads a configuration file and information about which PDB files to compare and then executes one of the actions as specified by the `-action` parameter. First, MAMMOTH is used to create a structural alignment of the receptors in the PDB files, placing the output in files with suffix `.aln`. These files are now read by the `create_expected_pairs` program which finds the closest pairs from the two different ligands and writes them to file. The `create` and `createm` run Superimpose and MultiBind on the binding sites specified in the `pdb_comparisons` file. The output is placed under the relevant `data/` directory. The `extract` action may be run after running the two programs to create output data. This creates a summary of the results obtained by the programs and places it in a database that may be queried using SQL. Finally the `query` action may be used to create summaries of the data in the database, including plots of graphs.

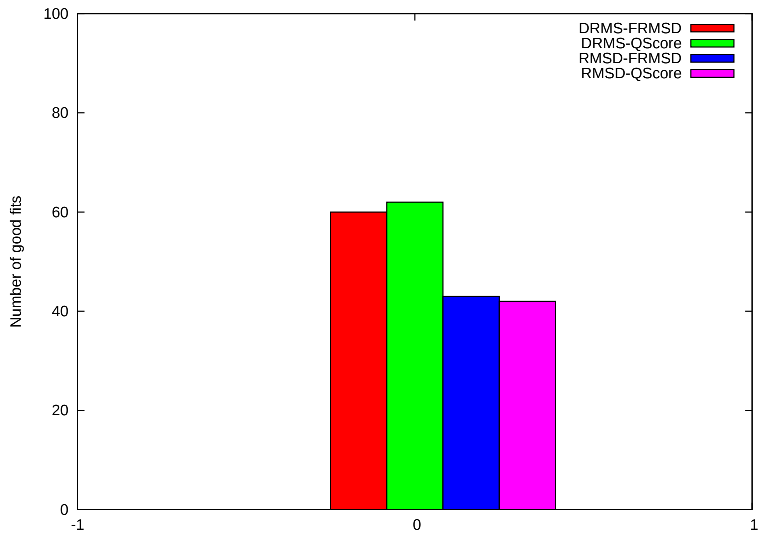


Figure 12: Total number of cases with 4 or more good pairs shown for each of the 4 combinations of outlier removal and fitness measure. The cases where the DRMS is used the number of good fits is significantly higher indicating that the DRMS measure is superior to the RMSD measure for determining the fitness. It is not possible to determine the best method of outlier removal from these results.

the binding site superimpositions where 4 or more of the expected pairs are found. Similarly the term *good pairs* refers to found pairs that are also expected pairs.

Taking cases where both MultiBind and Superimpose find good fits I looked at the distribution of the good fits of Superimpose into the 4 categories. The results can be seen in figure 12. Note that the results shown are across all of the lambda values tested.

The figure clearly shows that a larger amount of good fits are found in the runs where the DRMS fitness measure has been used.

Having established that using `DRMS` performs the best as fitness measure, I looked at which lambda values achieved the highest number of good fits. The same “filtering” as mentioned abova has been used. The results of this can be seen in figure 13.

In this figure the DRMS-QScore combination does well for lambda values 1.0, 2.0 and 4.0 with a somewhat surprising lack of good performance for lambda values of 3.0 and 5.0. I am unsure of the cause of the inconsistent performance of the cases where the DRMS fitness measure is used. More careful inspection of test results should be made to investigate this.

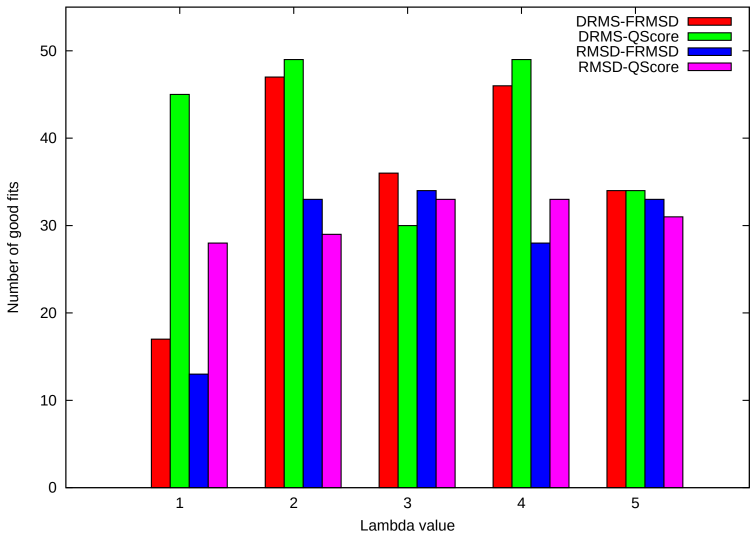


Figure 13: Distribution of results in the 4 categories across lambda values 1.0 - 5.0. The DRMS-QScore combination performs best with a lambda value of 1.0, 2.0 or 4.0.

In conclusion, we seem to get the best results by running Superimpose using the DRMS fitness method and the QScore method of outlier removal. There is an indication that some lambda values perform significantly better than others, but results are inconsistent. This indicates the need for further tests.

5.2.4 Comparing with MultiBind

The main motivation for comparing Superimpose and MultiBind was to have a comparison with an existing program that solves a similar problem, but uses a fundamentally different methodology that is well established.

In the paper presenting the MultiBind method [SPSNW08] the authors use a threshold of 4Å within the surface area to define the binding site that is used for the calculation of the superimposition. This is also the threshold value used in the comparison with Superimpose. The best settings of Superimpose were used to compare with, meaning the DRMS-QScore combination with a lambda value of 4.0.

To compare the two programs I looked at the *precision* and *recall* measures that are commonly used in information retrieval, as well as the *F-Measure* which is a combination of precision and recall.

The precision indicates how large a fraction of true positives are found, i.e. the fraction of found pairs that are good pairs. It is calculated as

$$precision = \frac{good\ pairs}{found\ pairs} \tag{9}$$

The recall indicates how large a fraction of the expected pairs we find. It is calculated as

$$recall = \frac{good\ pairs}{expected\ pairs} \tag{10}$$

The precision and recall can be combined into the F-measure which is a weighted mean of precision and recall. I have calculated it as

$$F = 2 * \frac{precision * recall}{precision + recall} \tag{11}$$

In the following graphs the precision and recall fractions have been converted into a percentage value.

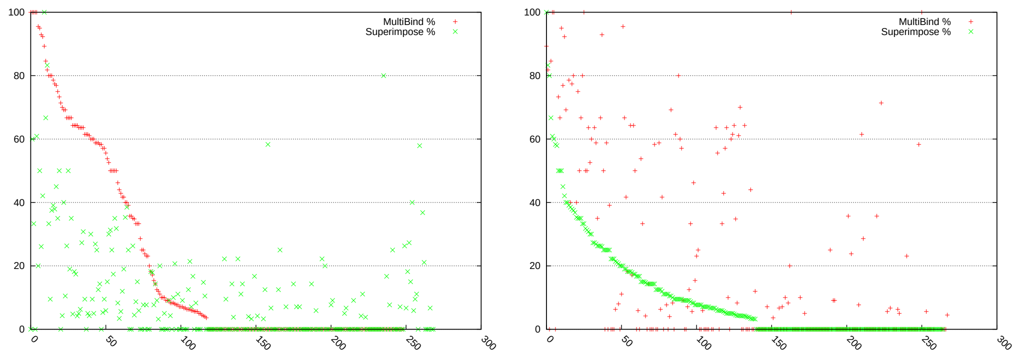


Figure 14: Percentage of *good pairs* of all found pairs. The graph on the left shows the results of MultiBind sorted in descending order. The results in right graph have been sorted by the precision values for Superimpose.

The following graphs all have a percentage or a fraction on the y-axis and an enumeration of the comparisons is on the x-axis.

In figure 14 the calculated precision values are shown. In the first graph the results have been sorted on the values for MultiBind in descending order. The second graph shows the same results only sorted by the precision values of results reported by Superimpose.

It can be seen that MultiBind has a higher precision up to around the first 80 comparisons where the second plateau in the left graph begins. MultiBind has a higher precision than Superimpose for all the cases to the point where the graph for MultiBind crosses 20%. After this point many of the precision values for Superimpose are higher, but only few have a precision higher than 25%.

The graphs show that there is a correlation between the cases where both Superimpose and MultiBind have a high precision. For the first 50 cases MultiBind reports a higher precision than Superimpose in most of the cases. After this there are some cases where Superimpose does better and some where MultiBind does better.

Also, Superimpose reports more cases with a positive precision, around 140 where MultiBind reports a positive precision for around the first 110 cases.

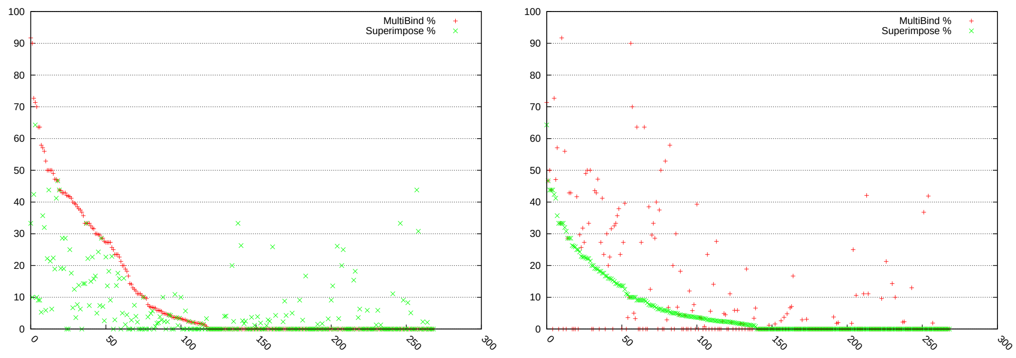


Figure 15: The percentage of the found pairs that are also in the list of expected pairs. The x-axis enumerates the compared cases. In the first graph the results have been sorted in descending order on the results of running MultiBind. The second graph shows the same data only sorted on the results from Superimpose.

In figure 15 the graph for MultiBind is somewhat similar. Only after around the 75 first comparisons cases are found where Superimpose does better than MultiBind.

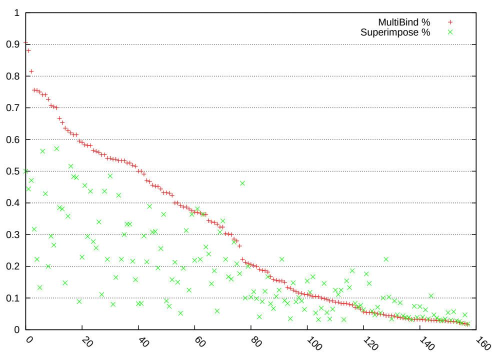
Looking at the right graph for Superimpose there seems to be a tendency that MultiBind has a higher recall than Superimpose or a recall of 0 where the recall of Superimpose is greater than 0. Cases in between are very few. This could indicate noise.

Again it is notable that Superimpose reports a positive recall for more comparisons than MultiBind does.

Figure 16 shows the F-measure calculated from the values used to produce the previous figures. Only cases where the F-measure is positive is included. MultiBind has a significantly higher F-measure up to around the 100 first comparisons. After this point the F-measure of the two programs does not differ significantly.

5.3 Discussion

The specific test cases shown in section 5.1.1 demonstrate that there is a need to reimplement how the weighting is done if Superimpose is to find



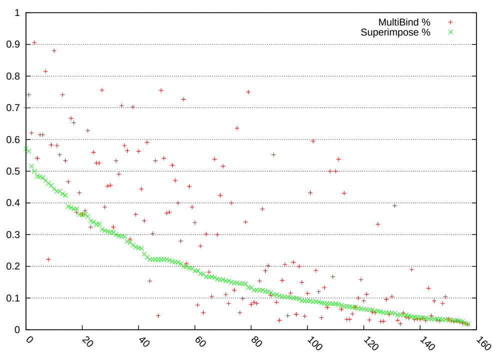


Figure 16: F-Measure values sorted

biologically interesting results. The result outcomes shown are rather good when compared to the tests performed on a larger extraction of data from SCOWLP. This is likely linked to the fact that the test cases in this section are from the same family, and are therefore structurally more similar than any of the cases in the later tests, where all superimposed binding sites are from different families.

It is established that Superimpose gives the best results when run with the DRMS method of determining the quality of a superimposition in section 5.2.3. This makes sense as the DRMS measure incorporates more information than the RMSD measure and also indicates that the fitness measure here is effective in changing which superimpositions the program will arrive at. To me this is also an indication that better biological results may be obtained by enhancing the fitness function to take more of the properties of the two binding sites into account.

The attempt at determining with which lambda value the best results can be obtained are unclear and there is a need for more work here to establish this.

From the test data presented in section 5.2.4 it can be seen that MultiBind performs better than Superimpose for most of the test cases. However, there seem to be some cases where Superimpose finds better results than MultiBind and these would be interesting to take a closer look at.

The way that pseudo atoms are defined by the two programs differs in such a way that MultiBind will report a larger amount of atoms than Superimpose for the same superimposition. This gives an advantage to MultiBind in that more pseudo atom pairs will be reported in the output. Also, the geometrical alignment is be expected to be better with a more detailed representation of binding sites.

Another possible source of MultiBinds superior performance on the test cases lies in the employment of a much more advanced weighting function than used by Superimpose.

There seems to be a fair amount of inaccuracy found in the test data and it is unsure exactly how this effects the results. This should be investigated.

The test method should also be evaluated further and possibly extended by also looking at which closes pairs that exist between the receptors of the two protein complexes.

6 Future directions

Here I will mention some possibilities of future development of the Superimpose program. Many of these ideas came up during the project, but were not pursued due to lack of time.

I believe that the most interesting further development of this program lies in bettering results by improving on the fitness function. This I would do by working on a few cases where MultiBind performs well, but Superimpose performs poorly.

6.1 Test data

The comparison of the two programs is unfair because the definition of pseudo atoms differs between the two programs. This should be changed and maybe a different approach to comparing superimposition quality should be made to avoid inaccurate results. A possible addition would be to include the comparison of closest pairs between the MAMMOTH aligned receptors when evaluating the quality.

6.1.1 Evaluation of the test

The comparison between MultiBind and Superimpose indicates that there is a fair amount of inaccuracy in the test data, which has also been verified to some extent by looking at some of the data output from MAMMOTH. This has not been done thoroughly, but more effort should be put into guaranteeing the quality of the data.

A possible start to do this would be to divide data into categories similar to those shown in figure 17 and pick representatives of each category to take a more careful look at.

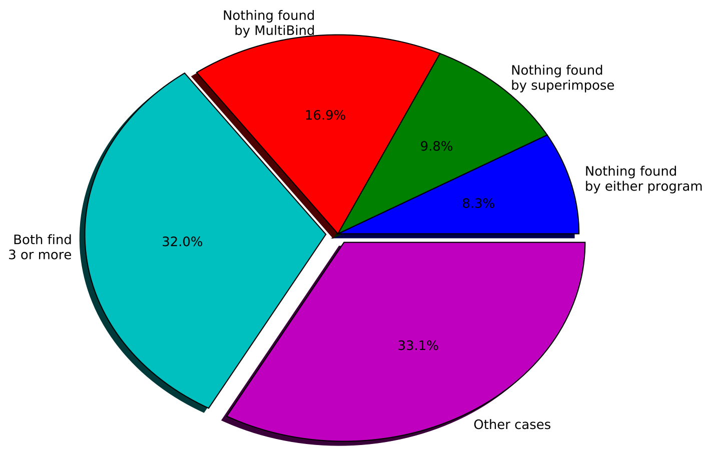


Figure 17: A rough division of data into 3 categories has been made. Representatives of each category should be visually inspected.

6.2 Weighting

The initial method of weighting does not function well. I believe this is because the weighting function is not placed correctly; It currently only affects the pairings of atoms.

The correct place to put a weighting function would be in the computation of the fitness value that is returned to each particle after ICP converges. This is also indicated by better performance of DRMS compared to the RMSD fitness method.

A place to start would be to move the existing weighting function away from its current location and reshape it into a fitness function. Initial steps to do this have been made in the file `fitness.h`.

The use of chemical properties should also be more nuanced. Disallowing an interaction between hydrophilic and hydrophobic atoms is too restrictive. Instead an analysis should be made of the properties between interacting atoms in the bindings we know from the PDB and this information should be used to create a better weighting function.

6.3 Artificial test data

An alternative to using real-world binding site data would be to create the artificial binding sites. This could be valuable in benchmarking Superimpose because features such as outliers and chemical properties may be varied systematically. This could give a more accurate estimation of how well the program performs with different kinds of data.

The program `make_random_sites` was written to be able to perform benchmarks like this, but the effort was discontinued in favour of a comparison with an existing program. However, the program is functional and systematic benchmarks would be easy to make with this.

The program creates two files containing a specified number of atoms distributed randomly within a specified distance from the origin. One file contains the Model and another the Data. The coordinates of Data file generated are translated a user specified amount in a random direction and also rotated a user specified number of degrees around a random axis. The user may also specify how many outliers there should be.

Initial tests of Superimpose on data generated with this program have been made, but unfortunately time constraints have not allowed inclusion of these results in this report.

However, the initial tests have shown that the perfect superimposition is found almost immediately if the two point sets have the same spacial configuration, (i.e. no outliers) but the Data point set is translated and rotated differently than the Model.

7 Conclusion

I have successfully implemented the Superimpose program, which uses a combination of the Particle Swarm Optimization and Iterative Closest Point algorithms to create a superimposition of the atoms of two protein binding regions.

Several methods for matching atom pairs and for removing atoms that are deemed to be irrelevant have been tried. There are many parameters to the program allowing a user to experiment with a variety of settings. The program has a highly modular design, making it open to future extensions.

The program has been tested by comparing the results of a manually created superimposition, with the results of running the program on the same input data and it has been compared with MultiBind on a larger data set extracted from SCOWLP. Results show that the initial weighting methodology implemented does not perform well and that the developed program can not compete with MultiBind at this point in time.

Tests to determine the best combination of fitness function and method of outlier removal have been made on a relevant data set. These tests indicate that there is room for further improvement of the fitness function.

References

[Ben90] Jon L. Bentley, *K-d trees for semidynamic point sets*, SCG '90: Proceedings of the sixth annual symposium on Computational geometry (New York, NY, USA), ACM, 1990, pp. 187–197. 15

[BM92] P. J. Besl and H. D. Mckay, *A method for registration of 3-d shapes*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **14** (1992), no. 2, 239–256. 7, 13

[BWF$^+$00] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne, *The RCSB Protein Data Bank*, Nucleic Acids Research (2000), 235–242, Online at http://www.pdb.org. 1

[CD01] Anthony Carlisle and Gerry Dozier, *An Off-The-Shelf-PSO*, 2001, pp. 1–6. 12

[CSK05] Dmitry Chetverikov, Dmitry Stepanov, and Pavel Krsek, *Robust euclidean alignment of 3d point sets: the trimmed iterative closest point algorithm*, Image and Vision Computing **23** (2005), no. 3, 299–309. 13

[EK95] R. Eberhart and J. Kennedy, *A new optimizer using particle swarm theory*, 1995, pp. 39–43. 7, 12

[GKM04] Susan L. Graham, Peter B. Kessler, and Marshall K. McKusick, *gprof: a call graph execution profiler*, SIGPLAN Not. **39** (2004), no. 4, 49–57. 21

[GS62] D. Gale and L. S. Shapley, *College Admissions and the Stability of Marriage*, The American Mathematical Monthly **69** (1962), no. 1, 9–15. 16

[gsl03] *Gnu scientific library: Reference manual*, Network Theory Ltd., February 2003, Online at http://www.gnu.org/software/gsl. 20

[HE02] Xiaohui Hu and Russell Eberhart, *Solving constrained nonlinear optimization problems with particle swarm optimization*, 6th World a comparison has been made withMulticonference on Systemics, Cybernetics and Informatics (SCI 2002), 2002, pp. 203–206. 12

[Ho09] Bosco K. Ho, *RMSD: Root Mean Square Deviation*, 2009, Online at http://boscoh.com/protein/rmsd-root-mean-square-deviation. 14

[htt09] http://www.sqlite.org, *SQLite Home Page*, 2009, Online at http://www.sqlite.org. 29

[Kab76] W. Kabsch, *A solution for the best rotation to relate two sets of vectors*, Acta Crystallographica Section A **32** (1976), no. 5, 922–923. 14

[KEHK04] Krissinel, E., Henrick, and K., *Secondary-structure matching (ssm), a new tool for fast protein structure alignment in three dimensions*, Acta Crystallographica Section D: Biological Crystallography **60** (2004), no. 1, 2256–2268. 17

[LLMO05] Dmitry Lupyan, Alejandra Leo-Macias, and Angel R. Ortiz, *A new progressive-iterative algorithm for multiple structure alignment*, Bioinformatics **21** (2005), no. 15, 3255–3263. 26

[MCA$^+$07] Alexey G. Murzin, John-Marc Chandonia, Antonina Andreeva, Dave Howorth, Loredana Lo Conte, Bartlett G. Ailey, Steven E. Brenner, Tim J. P. Hubbard, and Cyrus Chothia, *SCOP: Structural Classification of Proteins*, 2007, Online at http://scop.mrc-lmb.cam.ac.uk/scop. 3

[Mun57] J. Munkres, *Algorithms for the assignment and transportation problems*, Journal of the Society of Industrial and Applied Mathematics **5** (1957), no. 1, 32–38. 16

[NS07] Nicholas Nethercote and Julian Seward, *Valgrind: a framework for heavyweight dynamic binary instrumentation*, SIGPLAN Not. **42** (2007), no. 6, 89–100. 21

[PLT06] Jeff M. Phillips, Ran Liu, and Carlo Tomasi, *Outlier Robust ICP for Minimizing Fractional RMSD*, CoRR **abs/cs/0606098** (2006). 7, 8, 13, 14, 16, 17

[rcs09] rcsb.org, *The `.pdb` file format*, 2009, Online at http://www.wwpdb.org/documentation/format32/v3.2.html. 6, 11, 49

[SI09] Accelrys Software Inc., *Discovery Studio Visualizer*, 2009, Online at http://accelrys.com/products/discovery-studio/visualization/discovery-studio-visualizer.html. 22

[SL03] Fabian Schwarzer and Itay Lotan, *Approximation of protein structure for fast similarity measures*, RECOMB '03: Proceedings of the seventh annual international conference on Research in computational molecular biology (New York, NY, USA), ACM, 2003, pp. 267–276. 18

[SPSNW08] Alexandra Shulman-Peleg, Maxim Shatsky, Ruth Nussinov, and Haim J. J. Wolfson, *Multibind and mappis: webservers for multiple alignment of protein 3d-binding sites and their interactions.*, Nucleic Acids Res **36** (2008), 260–264. 5, 32

[Tey08] Joan Teyra, *Structural Characterization of Water, Ligands and Proteins*, 2008, Online at http://scowlp.org. 4

[Wea09] John Weaver, *Munkres code v2.*, 2009, Online at http://johnweaver.zxdevelopment.com/2007/05/22/munkres-code-v2/. 16

[WR97] H. J. Wolfson and I. Rigoutsos, *Geometric hashing: an overview*, Computational Science and Engineering, IEEE [see also Computing in Science & Engineering] **4** (1997), no. 4, 10–21. 5

A Getting and building the program

The source code of the program is available from the following location, or on the CD included with this report.

`http://bladre.dk/superimpose`

On this page a version of Boost and the GNU Scientific Library can also be found. These will need to be installed to be able to compile Superimpose. The program has been successfully built and tested using the library versions available from the above page.

The following steps describe how to set up a running version of the program developed in this report. It is assumed that the library dependencies will be installed in `$HOME/install` and that the source code of the project has been extracted to the folder `$HOME/superimpose`.

- Download the Boost libraries to the `$HOME/install` folder, unpack and install it:

```
wget http://bladre.dk/superimpose/boost_1_38_0.tar.bz2
tar -xvjf boost_1_38_0.tar.bz2
cd boost_1_38_0
./configure --prefix=$HOME/install
make && make install
```

- Download the GNU Scientific Library to the `$HOME/install` folder and unpack and install it

```
wget http://bladre.dk/superimpose/gsl-1.9.tar.gz
tar -xvzf gsl-1.9.tar.gz
cd gsl-1.9
./configure --prefix=$HOME/install
make && make install
```

- Download Superimpose and unpack it. Note that the package is rather large

```
wget http://bladre.dk/superimpose/superimpose-v760.tar.bz2
tar -xvjf superimpose-v412.tar.bz
cd superimpose
```

- To be able to link with the installed libraries the variable `LD_LIBRARY_PATH` needs to include the directory containing these:

```
export LD_LIBRARY_PATH=$HOME/install/lib:$LD_LIBRARY_PATH
```

- To get compilation flags needed for the GNU Scientific Library the `PATH` variable should contain the `bin` directory where the libraries are installed

```
export PATH=$HOME/install/bin:$PATH
```

- The `BASE` variable in the `env.sh` file should be modified to point to the root of the directory containing the source code of the project.
- Depending on which version of `g++` you are using the variable `PRGOPT` in the `make.conf` file may also need to be modified to be able to link with the `boost-program-options` library.
- Now enter the `src` directory and type one of the following commands to build the `superimpose` executable.

  `make` to build an executable that includes debugging information.

  `BUILD=1 make` to build an optimized version of the program.
- To create code documentation the `doxygen` program has been used. The documentation can be created by entering the `doc/` directory and typing `make`.

B Program Options

This section contains a more thorough description of the program options than printed when running:

```
./superimpose --help
```

Options may be specified at the command line or in the file .superimposerc in the users home directory. If an option appears both places, the command line option will override the option in the .superimposerc file.

All options have defaults, so it is not necessary to specify all options. Check ./superimpose –help to see the defaults.

B.1 General options

`-V` **or** `--version`

Print out the program version information.

`-h` **or** `--help`

Print out a summary of the program options.

`-s` **or** `--show`

When this option is specified the program will not run, but rather print out an overview of option settings. The same overview is printed in each output file produced by `superimpose` such that a given superimposition may be reproduced.

B.2 Input options

`-m` **or** `--model`

Specify which .pdb file should be used as the model.

`-d` **or** `--data`

Specify one or more .pdb files to be used as data site(s) to fit to the model. This option is actually superfluous as

```
./superimpose --model foo.pdb --data bar.pdb --data xxx.pdb
```

is equivalent to

```
./superimpose --model foo.pdb bar.pdb xxx.pdb
```

`-i` **or** `--input`

Specify from which source to input data into the program. At the moment this option has no meaning because the only way to input data into the program is by loading from .pdb files.

`-v` **or** `--visualize`

Specify a file containing locations of the files containning model and data binding site descriptions aswell as the .pdb files containing the full protein. When specifying this option the transformations performed on the site will also be perfomed on the full proteins and the output file will contain the model site, data site, the corresponding model protein and the corresponding data protein in that order. This option overrides the `--model` and `--data` options. The argument is a file which contains lines in the following format:

```
# MODEL <site file> <full pdb>;DATA <site file> <full pdb>
#
MODEL 1x11_PTB.pdb 1x11.pdb ; DATA 1shc_PTB.pdb 1shc.pdb
MODEL test.pdb testF.pdb ; DATA data.pdb dataF.pdb
```

It is not important if the `MODEL` or `DATA` entry appears first or second.

`-o` **or** `--outdir`

Specify a directory in which to place all files output from the program. The default is the current directory.

`-x` **or** `--prefix`

Specify a prefix to prepend to each output file written

`-f` **or** `--config_file`

Specify a file to read the program configuration from. It defaults to ${HOME}/.superimposerc. Options specified on the commandline will override options in the config file.

`-t` **or** `--initialize`

How to initialize the two protein binding sites before starting the PSO and ICP superimpositions. There are two possible arguments: `centroid` specifies to just move the binding sites so that their centroid is located at the center of the coordinate system. `align` specifies to also align the directional vector of the binding site with the `z-axis` and also to find the broadest part of the binding site and align this with the `x-axis`.

`-e` **or** `--seed`

Specify a value to use as seed for the random number generator used. This can be useful if one wants to recreate a run that is identical to a previous run. The default is to use the value returned by `time(2)`.

B.3 PSO specific options

`-p` **or** `--pso_particles`

Value that specifies how many particles to use in the swarm.

`-n` **or** `--pso_iterations`

Specifies how many iterations we should run the PSO before returning a result.

`-c` **or** `--pso_cognitive`

Specify the cognitive component weight in the formula used to update the velocity of each particle.

`-l` **or** `--pso_social`

This option is used to specify the weight of the social component in the formula used to update the particle velocities.

`-f` **or** `--pso_min_fitness`

If we arrive at a fitness value below this threshold the program stops and the result is written to the outfile.

`-w` **or** `--pso_wiggle`

This specifies the amount to "wiggle" a particle according to its own previous best position when we restart it. Each particle value is added a random number in the range -(wiggle/2) to (wiggle/2).

`-z` **or** `--pso_divisor`

This argument is used to set the divisor used to divide the initial velocities of the particles in the swarm.

B.4 ICP specific options

`-x` **or** `--icp_iterations`

Maximum number of iterations to run the ICP algorithm before returning a best fit.

`-y` **or** `--icp_min_points`

The minimum number of atoms that should be included in the fitting of data site to the model.

`-z` **or** `--icp_matching`

Specifies the method to use when performing a match between atoms of the data and model binding sites.

`-r` **or** `--icp_fraction_method`

> Specifies how to remove outliers from the matched points. Outliers will not be used in the calculation of the transformations to bring the data atoms closer to the model. There are some possibilities: `none` Do not remove outliers. `fixed` The argument to `--icp_lambda` specifies a fixed whole number that indicates exactly how many pairs of atoms from the matching to regard as inliers. `limit` Set a hard limit to cut away outliers. The value of `--icp_lambda` determines the allowed distance between atoms that separates inliers and outliers. `frmsd` Use the frmsd method of separating inliers and outliers. The value of `--icp_lambda` will be used to specify the lambda value as specified in the paper on the Fractional Iterative Closest Point algorithm. `qscore` Use the `qscore` function to determine where to separate inliers and outliers. The value of `--icp_lambda` will be used to set the divisor `R0` as used by the function.

`-b` **or** `--icp_lambda`

> This double value acts as argument to the chosen `--icp_-fraction_method`

`-A` **or** `--icp_max_angle`

> This option specifies the maximum angle displacement allowed between the model and data sites directional vectors. If a superimposition has a greater angle it is discarded.

`-W` **or** `--icp_weight`

> The method of weighting to apply. Currently options are: `none` specifies to not use weighting. `simple` A simple weighting to apply. Is currently not fully implemented.

`-X` **or** `--icp_weight_fraction`

> Specifies the fraction of the distance between two atoms that can be influenced by the `simple` `--icp_weight` method.

`-P` **or** `--icp_fixed_pairing`

> Specify a file from which to read information of a fixed pairing of atoms. This options is for refining the output of a manual superimposition by using only the ICP method. There will not be any use of the particle swarm with this option. When using this option the following settings are implied: `--icp_weight none`, `--initialize centroid`, `--icp_iterations 500`

`-F` **or** `--icp_fitness`

Specify which fitness method to use. Legal values are `rmsd` or `drms`. `rmsd` is the default.

C The modified .pdb file format

As input, Superimpose takes files in a format that is similar to that of `.pdb` files [rcs09] with some specific differences that have been added to accommodate our needs.

It was decided to use a similar format as that used by the PDB because this would allow us to easily produce data for testing because the PDB format is also used in other contexts.

The first difference to mention is that Superimpose uses only the `ATOM` and `HETATM` lines of the PDB. Additionally, the differences to the data in the columns of a file in the PDB format are as follows:

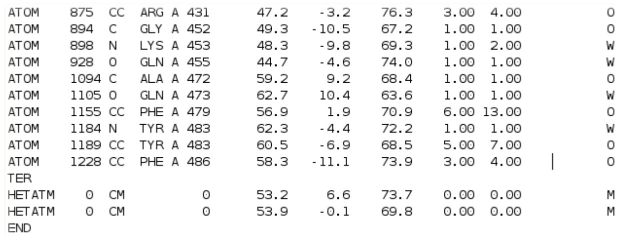
tempfactor In our format the tempfactor is a number defining the number of atoms that the given atom has been reduced to.

occupancy This column describes how many interactions there are for this atom.

These columns have been specified like this with the weighting of atoms in mind.

element In our format the element section describes if the atom is hydrophilic or hydrophobic. A `W` denotes a hydrophilic atom and an `O` a hydrophobic atom.

Finally, Superimpose expects that the `ATOM` section is terminated by a `TER` line and that a `HETATM` follows which holds the coordinates of the center of mass of the protein. This information is used together with the center of mass of the binding site to create a vector which is used to determine the directionality of the binding site.



```
ATOM    875  CC   ARG A 431      47.2    -3.2    76.3    3.00  4.00          O
ATOM    894  C    GLY A 452      49.3   -10.5    67.2    1.00  1.00          O
ATOM    898  N    LYS A 453      48.3    -9.8    69.3    1.00  2.00          W
ATOM    928  O    GLN A 455      44.7    -4.6    74.0    1.00  1.00          W
ATOM    1094 C    ALA A 472      59.2     9.2    68.4    1.00  1.00          O
ATOM    1105 O    GLN A 473      62.7    10.4    63.6    1.00  1.00          W
ATOM    1155 CC   PHE A 479      56.9     1.9    70.9    6.00 13.00          O
ATOM    1184 N    TYR A 483      62.3    -4.4    72.2    1.00  1.00          W
ATOM    1189 CC   TYR A 483      60.5    -6.9    68.5    5.00  7.00          O
ATOM    1228 CC   PHE A 486      58.3   -11.1    73.9    3.00  4.00    |     O
TER
HETATM    0  CM             0    53.2     6.6    73.7    0.00  0.00          M
HETATM    0  CM             0    53.9    -0.1    69.8    0.00  0.00          M
END
```

Figure 18: an example of the modified .pdb file format as found in the `.site` files. The last column denotes the hydrophobicity or hydrophilicity of the atom. The two columns before denote the number of atoms used to create the pseudo atom and the total number of interactions that the pseudo atom represents.

D Class description

The following are descriptions of the main classes and files that Superimpose consists of. To save some trees the program listing is not included, but the source code may be found at the following location:

`http://bladre.dk/superimpose`

D.1 atom

This class contains the description of atoms as used in the program. Class atom contains fields that are similar to the fields in the .pdb, although there are some variations. The description of the atom in the .pdb format can be found here:
`http://www.wwpdb.org/documentation/format32/sect9.html#ATOM`
The variations in our format are as follows:

- We do not use the occupancy and tempfactor fields for the same thing as described in the .pdb. Instead the occupancy field describes how many atoms have been used in averaging to get the atom. The tempfactor field we use to list how many interactions there are for this atom.
- The element column is used to denote if the atom can be considered to be hydrophilic or hydrophobic, denoted with `W` and `O` respectively.

D.2 config

The `config` class holds all configuration of the program. It uses Boost Program Options to parse commandline options, or to read settings form a configuration file.

Class config implements the Singleton pattern and is used mainly by the factory class for initializing newly created objects.

A macro CONF() is provided for easy access to configuration parameters.

D.3 constraints

This file contains the implementation of the constraints that are imposed on a resulting superimposition. The constraints include a maximum angle that there can be between the directional vector of the Model and Data sites and also the maximum displacement that is allowed for a sites centroid away from the center of the coordinate system.

The method `call()` returns a boolean, indicating if the constraints have been violated or not.

D.4 defines

This file contains constants and convenient macros that are used throughout the program.

Default values for configuration parameters are also found here.

D.5 factory

The `factory` class implements a variant of the factory pattern. All of the central objects in the program are created by the `factory` class. The `config` class is used heavily by the `factory` class.

All object creating methods in this class return a `std::auto_ptr` which assures that the memory occupied by the object will be deallocated at scope exit.

No instance of the `factory` class should be created, so the constructor is private.

D.6 fitness

This file defines classes implementing the RMSD and dRMS fitness measures.

D.7 fraction

The `fraction` class is the superclass for specifying how to separate inliers and outliers from the sorted matching list. For most of these the `-icp_-lambda` option declares a parameter to be passed.

`fraction_none` Do not compute outliers, but regard everything as inlies

`fraction_limit` is used for setting a hard limit on allowed distance between 2 atoms

`fraction_fixed` Is used to set a fixed number of pairs to always be considered as inliers

`fraction_frmsd` Uses the FRMSD method for determining which are outliers.

`fraction_qscore` This uses the q-score for calculating the boundary between inliers and outliers.

D.8 icp

This class implements a variant of the Iterative Closest Point algorithm. It uses the implementation of the Kabsch algorithm as found in rmsd.c

The class uses an object of type `matching` to perform the pairing between points. Class `fraction` is used to determine which pairs should be regarded as inliers and which should be regarded as outliers.

D.9 initializer

The `initializer` class is used to initialize a binding site so that all atoms in the site are moved to the center of the coordinate system by placing them such that the centroid of the protein is in the center of the coordinate system.

There are two kinds of initializers: `initializer_centroid` simply moves the points in the protein so that the protein centroid is at the center of the coordinate system. `initializer_align` additionally aligns the directional vector of the site with the `z-axis` and also aligns the broadest part of the binding site with the `x-axis`.

D.10 loader

This file contains classes for loading data into the program. The classes are:

`loader` is an abstract class with some methods for loading files in our special .pdb format.

`loader_file` An instance of this class will be used to load .pdb file data from files.

`loader_visualize` This loader is used when we are getting data from a file specified with the `-visualize` option.

`fixed_pairing_loader` is used to load a file with two columns of the atom serial number describing which pairs of atoms to fix.

`visualize_file_loader` is used to load the file containing model and data binding sites to load and also a reference to the full .pdb file so that the rotations and translations applied to the binding sites can be seen in the context of the whole protein. It is an auxilliary class used by `loader_-visualize`.

D.11 matching

This file contains classes that implement different approaches to performing a pairing between atoms from two different binding sites.

The name matching has been preferred to pairing when describing the process of finding a match for an atom. The word pairing is used to describe a matching that has been performed.

There are different approaches to matching. These are implemented in different sub classes:

`match` is the abstract base class for performing matchings between the atoms of two sites.

`match_fixed` is used to implement a fixed matching that must be specified by the user in a file given as argument to the `-icp_fixed_-pairing` option.

`match_munkres` performs the matching by first computing a grid of all distances between atoms of the two sites, and then uses the Munkres algorithm to find atom pairs such that all atoms have a pair and the summed distance between atom pairs is minimized.

`match_simple` This is a simple variant of the approach of computing a grid without using the Munkres algorithm. Not all atoms will have a corresponding pair in all cases when using this approach, but it is much faster. The method is rather ad hoc.

D.12 randomXX

This class is just a wrapper around the random number generators that are provided by the GNU Scientific Library.

D.13 scene

In the scene class all objects are put together and the actual superimposition is started by executing the `run()` method.

D.14 site

The `site` class is used to describe a binding site as loaded by the `loader` class. All transformations done to a site are done through the `move()` methods.

When the `-visualize` option has been specified, invoking `move()` will also perform the same transformation to the full site.

The `site` class contains all information about a binding site, including atoms and the center of the protein and the center of the binding site.

D.15 swarm

Implements the Particle Swarm and is the main entry point for the whole machinery in the program.

This file contains the following classes:

`particle` is a class implementing the particles that are used by the swarm class.

`swarm` is an abstract class describing the interface for the two different "swarms".

`swarm_iterate` implements the Particle Swarm Optimization. It uses the `icp` implementation to get a fitness value for determining the fitness of a particle.

`swarm_only_icp` is a dummy Particle Swarm which is used in the case where one wishes to optimize an input using only the `icp` algorithm.

D.16 utils

This file contains general utility functions that are used throughout the program and have not been associated well with a class.

D.17 vec

The file `vec.h` contains an implementation of 3D vectors and matrixes used, is found in this file. It is a convenience for being able to do basic transformations in a nice way.

D.18 weights

Contains the basic implementation of weights as used in the program. The basic method of weighting is to increase or decrease the distance between atom pairs in a matching according to the number of interactions for the two atoms and also the number of atoms that have gone into the atom when averaging it. Besides the abstract class `weight` there are the following sub classes :

`weight_none` This is a dummy class that is used when no weighting is applied to a matching.

`weight_simple` is an implemntation of a weighting that can be applied for modifying the distance between an atom pair.

`weight_better` is an attempt to try another method of weighting. It is currently not used.

The method of weighting atom pairings should be extended in future work.

D.19 writer

The `writer` class is a base class for classes implementing the writing of the resulting superimposition. Currently this includes only writing output to a .pdb file which follows our internal variant of the .pdb. This is convenient because the resulting output can be loaded into programs that can be used to visualize the superimposition.

When the `-visualize` option is specified the file written will contain the different parts in the following order:

- Model binding site
- Data binding site
- Model "full" site
- Data "full" site Each separated with a TER