MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Superimposing Multiple Structures and Exploring Protein Binding Sites

MASTER'S THESIS

David Sehnal

Brno, Autumn 2009

Declaration

I hereby declare that this paper is my original authorial work, which I have worked out on my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: RNDr. Radka Svobodová Vařeková, PhD.

Acknowledgement

I would like to thank my supervisor RNDr. Radka Svobodová Vařeková, PhD. for introducing me to the topic and the leadership throughout writing this master's thesis. Also, I would like to thank to Lukáš Pravda and Jan Oppelt for testing the software described in this thesis, and Mgr. Lenka Zalabová, PhD. and Ing. Crina-Maria Ionescu for their feedback. Last but not least, I would like to thank my family and Tereza Pospíšilová for their support.

The access to the MetaCentrum computing facilities provided under the research intent MSM6383917201 is appreciated.

Abstract

This master's thesis is focused on superimposing molecular structures, particularly protein binding sites. Three algorithms are presented: the first for superimposing a pair of three-dimensional structures, the second for finding a pairing of atoms in protein structures, and, finally, the third for superimposing multiple structures at once. Furthermore, models of particular binding sites can be computed using the algorithm for multiple alignment. Part of the thesis is the implementation of the algorithms in a software package called SiteBinder, that also provides a user friendly graphical interface. SiteBinder was successfully tested on real data and compared to other available software for superimposing molecular structures. The results provided by SiteBinder were better or at least as good as those provided by the other software packages.

Keywords

quaternion, rotation matrix, RMSD, multiple, superimposition, alignment, protein, binding site, SiteBinder

Contents

Part 1

Introduction

CHAPTER 1

Introduction

During the last couple of decades many interdisciplinary fields at the edge of natural sciences (especially chemistry, biology, and genetics) and computer science flourished. To name a few: computational chemistry, molecular modeling, bioinformatics, systems biology, and cheminformatics. At first, the goal of these disciplines was to substitute the real world experiments by a simulation, but in time the focus has somewhat broadened. Therefore, one of the new directions of research is the analysis and processing of large volumes of structural molecular data (proteins, nucleic acids, etc.).

The last mentioned field is developing rapidly especially thanks to the modern methods of biomolecule structural analysis that provide very large volumes of data. Bioinformatics is focused on studying (bio)molecules, especially on features relevant to molecular biology and genetics. Cheminformatics provides us with a very interesting and useful point of view on biomolecules that mainly focuses, as opposed to bioinformatics, on particular parts (motifs) of biomolecules (for example active and binding sites [**1, 2**]).

An important building and functional fabric of cells of living organisms consists of proteins [**3**]. These are large molecules (containing up to thousands of atoms), but only very small subparts of them (the so called active sites) take part in chemical reactions. Some proteins contain atoms of metals (the so called metalloproteins [**4**]), and it is these metals' biding sites that are often the active sites of proteins. Comparing the structure of different binding sites of a given metal can give very valuable information about that particular site. One example is describing how the geometry differs in various organisms or how the shape of the binding site relates to the activity and chemical behavior of molecules. By superimposing (structurally aligning) the binding sites (so that the difference between the superimposed binding sites is as small as possible) one can also obtain a model of the binding site by computing an average structure. This model can be used to predict a binding site in molecules that released the particular metals during structural analysis (because the metals are bound by a weaker bond than other atoms).

The aim of this thesis is to create a software package for superimposing a large number of structural motifs, most notably the binding sites of metals in proteins. The superimposition is achieved by minimizing the value of RMSD [**5**] (root mean square deviation – the average distance of corresponding atoms) of the participating structures. The key factor when superimposing molecules is the fact that the pairing of atoms of all participating structures must be established. In some cases, the pairing is given a priori, nevertheless, in general, this is not so. Therefore, the problem of finding an isomorphism between protein structures (i.e. pairing of their atoms) is also addressed in this thesis. Ultimately, once the structures are superimposed, the average structure is computed that serves as model of the particular motif.

This master's thesis was completed in Computational chemistry group at National Centre for Biomolecular Research, Masaryk university, Brno. The thesis is a part of the research topic Metalloproteins and metalloproteinic sensors, and it was created in cooperation with the ANF DATA, spol s r.o. (the Siemens company), specifically with the Siemens CT T CEE Life Science Systems.

1. Goals of the Thesis

The goals of my thesis are these:

- Study/develop algorithms for superimposing two and more molecules (here the work from the author's bachelor thesis [**6**] is continued).

- Develop and describe algorithms for finding optimal pairing of atoms and superimposing (multiple) molecular structures.
- Implement software that loads and displays structural motifs. Furthermore, the software should allow the selection of several displayed motifs, their superimposition, computation of RMSD, and computation of the average structure.
- Test the software on real binding sites, obtained from the Protein Data Bank database.

Part 2

Theory

CHAPTER 2

Introduction to Chemical Background

1. Fundamental Chemical Terms

The chemical terms described in the following text are the counterparts of the abstract notions used throughout the thesis.

Atom: An atom [**7**] is the smallest possible particle of a chemical element that retains its chemical properties. It consists of a positively charged nucleus surrounded by one or more negative electrons.

Chemical bond: A chemical bond [**8**] is an interaction between atoms or molecules and allows the formation of polyatomic chemical compounds. It is the attraction caused by the electromagnetic force between opposing charges, either between electrons and nuclei, or as the result of a dipole attraction. The strength of bonds varies considerably; there are "strong bonds" such as covalent or ionic bonds and "weak bonds" such as dipole-dipole interactions, the London dispersion force and hydrogen bonding.

Molecule: A molecule [**9**] is defined as an electrically neutral group of at least two atoms in a definite arrangement held together by very strong (covalent) chemical bonds.

Amino acid: Amino acids [**10**] are molecules containing an amine group ($-NH_2$), a carboxylic acid group ($-COOH$) and a side chain that varies between different amino acids. These molecules are particularly important in biochemistry, where this term refers to alpha-amino acids with the general formula $NH_2 - CHR - COOH$, where R is an organic substituent. In the alpha-amino acids, the amino and carboxylic groups are attached to the same carbon atom, which is called the alpha carbon. The various alpha amino acids differ in which side chain (R group) is attached to their alpha carbon. These side chains can vary in size from just a hydrogen atom in glycine, to a methyl group in alanine, through to a large heterocyclic group in tryptophan.

Protein: Proteins [**3**] (also known as polypeptides) are organic compounds made of amino acids arranged in a linear chain and folded into a globular form. The amino acids in a polymer are joined together by the peptide bonds between the carboxyl and amino groups of adjacent amino acid residues.

Amino acid residue: An amino acid residue [**11**] is what is left of an amino acid once a molecule of water has been lost (an H^+ from the amine group and an OH^- from the carboxylic group) in the formation of a peptide bond.

2. Protein Structure

A structure of a protein [**12**] (the organization of amino acids in a protein and the distribution of protein atoms in space) can be described on several different levels of abstraction.

Primary Protein Structure: The primary protein structure is the sequence of amino acids in a protein. It is customary to write the primary structure of a protein as a sequence of the one-letter (or three-letter) abbreviations of each amino acid in the chain.

Secondary Protein Structure: Within long proteins, certain sections fold into sheets or twist into coils. These regions with specific structural characteristics make up the secondary structure of the protein. The two most typical secondary structures within proteins are α-helix and β-sheet.

Tertiary Protein Structure: The tertiary protein structure is the result of the bends and folds that a protein chain adopts to achieve a shape, required for its biological function. It can be described by the Cartesian coordinates of all atoms of a molecule in a space.

Quaternary Protein Structure: The quaternary protein structure is the arrangement of multiple folded protein molecules in a multi-subunit complex.

Binding Site: A binding site [**2**] is a region on a protein, DNA, or RNA to which specific other molecules and ions in this context collectively called ligands, or more specifically, protein ligands form a chemical bond.

CHAPTER 3

Quaternions

This chapter gives a brief overview (taken and modified from the author's bachelor thesis [**6**]) of quaternion algebra necessary to understand the algorithm for finding the optimal alignment of three-dimensional structures presented in this thesis. This chapter is included because, unlike other mathematical tools used in the algorithm such as eigenvectors and eigenvalues, quaternions are not considered a "standard" tool.

1. Defining Quaternions

There are many ways a quaternion [**13**] can be looked at. These include a vector with four components, members of a non-commutative division algebra, composite of a scalar and an ordinary 3D vector (so in effect a 4D vector) or a hyper-complex number with three different imaginary parts. In this thesis we will usually use the hyper-complex number and vector representations of quaternions.

Definition 3.1. *A quaternion in the hyper-complex number notation is a number*

$$a = a_0 + a_x\mathbf{i} + a_y\mathbf{j} + a_z\mathbf{k}$$

where $a_0 \in \mathbb{R}$ is the real part and $a_x, a_y, a_z \in \mathbb{R}$ are imaginary parts for which

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1.$$

It can be shown, that the following holds for imaginary parts of quaternions:

$$\begin{array}{lll} \mathbf{ij} = \mathbf{k}, & \mathbf{jk} = \mathbf{i}, & \mathbf{ki} = \mathbf{j}, \\ \mathbf{ji} = -\mathbf{k}, & \mathbf{kj} = -\mathbf{i}, & \mathbf{ik} = -\mathbf{j}. \end{array}$$

Definition 3.2. *A quaternion in the vector notation is a vector $a = (a_0, a_x, a_y, a_z)^T \in \mathbb{R}^4$.*

2. Basic Operations

In the following let $a = a_0 + a_x\mathbf{i} + a_y\mathbf{j} + a_z\mathbf{k}$ and $b = b_0 + b_x\mathbf{i} + b_y\mathbf{j} + b_z\mathbf{k}$ be quaternions.

Definition 3.3. *The addition of two quaternions a and b is a quaternion*

$$a + b = (a_0 + b_0) + (a_x + b_x)\mathbf{i} + (a_y + b_y)\mathbf{j} + (a_z + b_z)\mathbf{k}.$$

Definition 3.4. *The multiplication of two quaternions a and b is a quaternion*

$$\begin{aligned} ab \;=\; & (a_0b_0 - a_xb_x - a_yb_y - a_zb_z) \\ & + (a_0b_x + a_xb_0 + a_yb_z - a_zb_y)\mathbf{i} \\ & + (a_0b_y - a_xb_z + a_yb_0 + a_zb_x)\mathbf{j} \\ & + (a_0b_z + a_xb_y - a_yb_x + a_zb_0)\mathbf{k}. \end{aligned}$$

Lemma 3.1. *The quaternion multiplication is non-commutative, therefore $ab \neq ba$.*

PROOF. A proof of this lemma is a straightforward application of the previous definition. □

Definition 3.5. *The multiplication of a quaternion a by a scalar s is a quaternion*

$$sa = sa_0 + sa_x\mathbf{i} + sa_y\mathbf{j} + sa_z\mathbf{k}.$$

Sometimes, it is convenient to express a product of two quaternions in terms of the product of a 4×4 matrix and a vector. We can choose to represent the left or the right multiplication as a matrix as shown in the following theorem.

Definition 3.6. *Let a, b be quaternions. Then*

$$ab = \begin{pmatrix} a_0 & -a_x & -a_y & -a_z \\ a_x & a_0 & -a_z & a_y \\ a_y & a_z & a_0 & -a_x \\ a_z & -a_y & a_x & a_0 \end{pmatrix} b = \mathbf{A}b$$

and

$$ba = \begin{pmatrix} a_0 & -a_x & -a_y & -a_z \\ a_x & a_0 & a_z & -a_y \\ a_y & -a_z & a_0 & a_x \\ a_z & a_y & -a_x & a_0 \end{pmatrix} b = \hat{\mathbf{A}}b.$$

Notice that $\hat{\mathbf{A}}$ differs from $\mathbf{A}$ in that the lower-right 3x3 sub-matrix is transposed which further illustrates the non-commutativity of quaternion multiplication. Moreover, the sum of the squares of elements of each column (or row) of $\mathbf{A}$ and $\hat{\mathbf{A}}$ is equal to

$$a_0^2 + a_x^2 + a_y^2 + a_z^2$$

which is the dot product $a \cdot a$ (or a square magnitude), defined below.

The matrix $\mathbf{A}$ ($\hat{\mathbf{A}}$) is obviously orthogonal (orthonormal if the quaternion has unit norm, defined below) therefore the matrix product $\mathbf{A}^T\mathbf{A} = (a \cdot a)\mathbf{I}$ is a diagonal matrix, where $\mathbf{I}$ is the identity matrix.

The dot product of two quaternions is defined as the sum of products of the corresponding components:

Definition 3.7. *A dot product of quaternions a and b is a real number*

$$a \cdot b = a_0 b_0 + a_x b_x + a_y b_y + a_z b_z.$$

Furthermore, we define a conjugate $(\cdot)^*$ of a quaternion.

Definition 3.8. *The conjugate of a quaternion a is a quaternion*

$$a^* = a_0 - a_x\mathbf{i} - a_y\mathbf{j} - a_z\mathbf{k}.$$

The 4×4 matrices associated with the conjugate of a quaternion are just the transposes of the matrices associated with the quaternion itself.

Definition 3.9. *The magnitude (norm) $\|\cdot\| \in \mathbb{R}$ of a quaternion is*

$$\|a\| = \sqrt{aa^*} = \sqrt{a_0^2 + a_x^2 + a_y^2 + a_z^2}.$$

Definition 3.10. *A unit quaternion is a quaternion whose magnitude is equal to 1.*

The inverse of a non-zero quaternion is defined in the same way as the inverse of a complex number.

Definition 3.11. *The inverse of a quaternion a is a quaternion*

$$a^{-1} = \frac{1}{a \cdot a^*} a^*.$$

3. Useful Identities

The following lemmas show some interesting properties of quaternion multiplication and dot product. Proofs can be found in **[14]**.

Lemma 3.2. *Let a, b, c be quaternions. Then*

$$(ab) \cdot (ac) = (\mathbf{A}b) \cdot (\mathbf{A}c) = (\mathbf{A}b)^T(\mathbf{A}c).$$

Lemma 3.3. *Let a, b, c be quaternions. Then*

$$(\mathbf{A}b)^T(\mathbf{A}c) = b^T\mathbf{A}^T\mathbf{A}c = b^T(a \cdot a)\mathbf{I}c.$$

From 3.2 and 3.3 we immediately get

Lemma 3.4. *Let a, b, c be quaternions. Then*

$$(ab) \cdot (ac) = (a \cdot a)(b \cdot c).$$

Extending lemma 3.4 further, in the case that a is an unit quaternion we obtain $(ab) \cdot (ac) = b \cdot c$. Moreover, in case $b = c$ we have

$$(ab) \cdot (ab) = (a \cdot a)(b \cdot b).$$

In other words, a (squared) magnitude of a product is the product of the magnitudes.

Another useful property of quaternions is

Lemma 3.5. *Let a, b, c be quaternions. Then*

$$(ab) \cdot c = a \cdot (bc^*).$$

4. Three-dimensional Vectors as Quaternions

3D vectors can be represented as purely imaginary quaternions.

Definition 3.12. *Let $r = (r_x, r_y, r_z)^T$ be a (column) vector. Then the corresponding quaternion is*

$$r = r_x\mathbf{i} + r_y\mathbf{j} + r_z\mathbf{k}.$$

Similarly, we can represent a scalar as a "real" quaternion.

Notice that the left and right multiplication matrices associated with a purely imaginary quaternion satisfy

$$\mathbf{R}^T = -\mathbf{R}$$

and

$$\hat{\mathbf{R}}^T = -\hat{\mathbf{R}}.$$

5. Rotations as Quaternion

There is a direct relation between unit quaternions and proper rotations in $\mathbb{R}^3$ **[13]**.

Definition 3.13. *A rotation by an angle θ about an axis represented by a unit vector $v = (v_x, v_y, v_z)^T$ is represented by a unit quaternion*

$$rq(\theta, v) = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}).$$

To transform a vector r we use the following:

Definition 3.14. *Let $r = r_x\mathbf{i} + r_y\mathbf{j} + r_z\mathbf{k}$ be a vector expressed as a purely imaginary quaternion. The rotation of vector r by the angle θ about the axis v is defined as*

$$r' = qrq^*,$$

where $q = rq(\theta, v)$.

The composition of two rotations is the product of the two corresponding quaternions.

Lemma 3.6. *Let q be a unit quaternion. Then the quaternion $-q$ represents the same rotation.*

PROOF. Indeed,

$$(-q)r(-q)^* = qrq^*.$$

□

Theorem 3.1. *A 3x3 rotation matrix corresponding to a rotation quaternion q is a matrix*

$$\begin{pmatrix} q_0^2+q_x^2-q_y^2-q_z^2 & 2(q_xq_y+q_0q_z) & 2(q_xq_z-q_0q_y) \\ 2(q_xq_y-q_0q_z) & q_0^2-q_x^2+q_y^2-q_z^2 & 2(q_yq_z-q_0q_x) \\ 2(q_xq_z+q_0q_y) & 2(q_yq_z-q_0q_x) & q_0^2-q_x^2-q_y^2+q_z^2 \end{pmatrix}.$$

PROOF. Expressing

$$qrq^*$$

from definition 3.14 as a matrix multiplication we get

$$\mathbf{Q}\hat{\mathbf{Q}}^* r = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & q_0^2+q_x^2-q_y^2-q_z^2 & 2(q_xq_y+q_0q_z) & 2(q_xq_z-q_0q_y) \\ 0 & 2(q_xq_y-q_0q_z) & q_0^2-q_x^2+q_y^2-q_z^2 & 2(q_yq_z-q_0q_x) \\ 0 & 2(q_xq_z+q_0q_y) & 2(q_yq_z-q_0q_x) & q_0^2-q_x^2-q_y^2+q_z^2 \end{pmatrix} r.$$

Finally, matrix $\mathbf{Q}\hat{\mathbf{Q}}^*$ acts only on the imaginary part of r. □

CHAPTER 4

Introduction to Superimposing of Structures

In the following three chapters we will describe three algorithms for superimposing structures. This chapter gives a brief overview of the problem and basic definitions and theorems used in the following text.

First, a method for superimposing two arbitrary 3D structures is described (i.e. finding a transformation that superimposes one structure onto another, minimizing the value of RMSD – the average distance between elements of the structures, defined below), assuming a paring of *atoms* in these structures is given a priori. Unfortunately, we do not always have the pairing available, therefore, in the next chapter, the algorithm is extended to search through automorphisms of the molecule graph of one of the structures in order to find the best pairing – one that yields the best superimposition. Finally, an algorithm for superimposing multiple structures is introduced.

This thesis is most concerned with the structure of metal binding sites – that means all the definitions, notations, etc., will assume that the underlying structure is a protein. Nevertheless, the presented algorithms can be directly extended to work for arbitrary molecular structures.

So, as hinted above, there are two main issues when dealing with superimposing two structures:

(1) A pairing of atoms must be established. The goal is to find a pairing that yields the best fit (i.e. the RMSD of the pair is the lowest).
(2) A transformation must be found that superimposes the two structures with given pairing.

The naive approach to the first problem would be to simply check all possible pairings of atoms resulting in $O(n!)$ time complexity (number of bijections of two sets of size n). This is of course very undesirable. An improvement would be to pair only sulfur atoms to sulfur atoms etc. But it turns out that we can do better.

Fortunately, we might take advantage of the structure of proteins (or the binding sites to be more specific) to, in some cases, dramatically reduce the search space. We discuss various approaches to this problem and an algorithm for searching the pairing space is presented and possible extensions are discussed.

Next, an algorithm for finding a transformation that superimposes two three-dimensional structures is presented. The algorithm is a mix of the work of **[14]** and **[15]** and builds upon the work done in the author's bachelor thesis **[6]**.

Ultimately, an algorithm based on the work in **[16]** for superimposing multiple structures is presented. A generalized version of RMSD is used to establish the quality of the given alignment.

Finally, we are interested only in small structural motifs (i.e. binding sites) of molecules. Therefore, when superimposing molecules we are generally not interested in the whole structure. Rather, only a selection of atoms will be relevant to the computed value of RMSD.

1. Basic Notations and Definitions

The basic element we will be dealing with in this thesis is *structure*, a mathematical object that represents the 3D structure of a protein (or its part). The *bonds* of atoms in molecules are ignored because there is no use for them in the presented algorithms. As mentioned at the beginning of this chapter, when aligning structures, we will usually be interested only in a subset of the atoms of the given structure.

Residues will play an important role in designing the algorithm for finding the optimal pairing of atoms. In proteins, residues usually correspond to individual amino acids.

Definition 4.1. *A structure is a nine-tuple* $S = (\mathrm{Ids}, \mathrm{Sym}, \mathrm{Sel}, \mathrm{RIds}, \mathrm{Res}, \beta, \tau, \rho, \pi)$ *where*

$\mathrm{Ids} \subset \mathbb{N}$**:** *An ordered (finite) set of atom identifiers. The order is arbitrary but fixed.*

Sym**:** *A set of element symbols, i.e.* $\{S, N, O\}$.

$\mathrm{Sel} \subset \mathrm{Ids}$**:** *(Ordered) selection of atoms. Throughout the text, the selection of atoms will often be implicitly denoted by a list of element symbols (i.e.* $\{S, C, C, N, N\}$*).*

$\mathrm{RIds} \subset \mathbb{N}$**:** *A (finite) set of residue identifiers.*

Res**:** *A set of residue names. Usually an abbreviation of the amino acid name (i.e. CYS for cysteine, HIS for histamine, etc.), however, any name is applicable (such as UNK for unknown, etc.).*

$\beta : \mathrm{Ids} \to \mathrm{Sym}$**:** *A mapping that assigns an element symbol to each atom in* Ids.

$\tau : \mathrm{Ids} \to \mathrm{RIds}$**:** *A mapping that assigns a residue identifier to every atom in* Ids.

$\rho : \mathrm{RIds} \to \mathrm{Res}$**:** *A mapping that assigns a residue name to every residue in* $RIds$.

$\pi : \mathrm{Ids} \to \mathbb{R}^3$**:** *A mapping that assigns a position to every atom. The unit used is Ångström (*Å*,* 1 Å$= 10^{-10}m$*).*

We denote the set of all structures by $\mathcal{S}$.
For a structure $S \in \mathcal{S}$ *we denote* $S_i = \pi(Sel_i)$ *as the position* i*-th atom of* Sel.

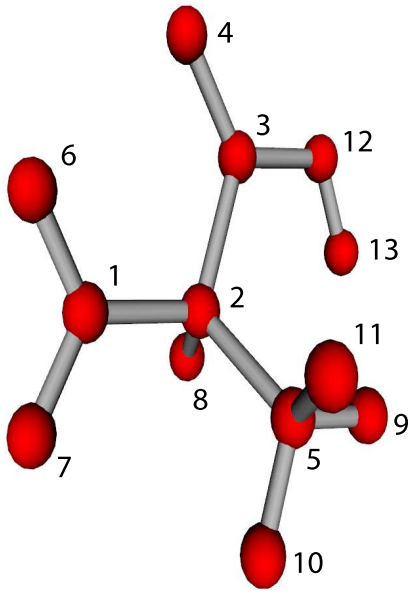


FIGURE 4.1. Alanine molecule.

Example 4.1. *This example shows the representation of the amino acid alanine (depicted in figure 4.1) in the* structure *formalism.*

Ala $= (\mathrm{Ids}, \mathrm{Sym}, \mathrm{Sel}, \mathrm{RIds}, \mathrm{Res}, \beta, \tau, \rho, \pi)$ *where*

$\mathrm{Ids} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$.

$\mathrm{Sym} = \{C, O, N, H\}$.

Sel *is arbitrary.*

$\mathrm{RIds} = \{1\}$.
$\mathrm{Res} = \{ALA\}$.
$\beta : 1 \mapsto N, 2 \mapsto C, 3 \mapsto C, 4 \mapsto O, 5 \mapsto C, 6 \mapsto H, 7 \mapsto H, 8 \mapsto H, 9 \mapsto H, 10 \mapsto H, 11 \mapsto H, 12 \mapsto O, 13 \mapsto H$.
$\tau : i \mapsto 1$, *where* $i = 1..13$.
$\rho : 1 \mapsto ALA$.
$\pi : 1 \mapsto (0.039, -0.028, 0.012), 2 \mapsto (1.499, -0.043, 0.012)$, *etc.*

Two structures are called *compatible* if their selections contain the same number of atoms and the i-th atoms have the same element symbol.

Definition 4.2. *We say that structures $A, B \in \mathcal{S}$ are compatible iff $|\mathrm{Sel}_A| = |\mathrm{Sel}_B|$ and $\beta_A((\mathrm{Sel}_A)_i) = \beta_B((\mathrm{Sel}_B)_i)$ for all suitable i.*

In order to measure how different two structures are, we use the root mean square deviation (RMSD), the average distance between corresponding atoms. We assume that both structures going into the formula are compatible (in the sense of definition 4.2), otherwise, the value of RMSD is not defined.

Definition 4.3. *We define function* $\mathrm{RMSD} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ *as*

$$\mathrm{RMSD}(A, B) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \|A_i - B_i\|^2},$$

where $A, B \in \mathcal{S}$ are compatible structures, $n = |\mathrm{Sel}_A| = |\mathrm{Sel}_B|$ and $\|v\| = \sqrt{v_x^2 + v_y^2 + v_z^2}$ is the standard vector norm.

Now, we extend the value of RMSD to an arbitrary number of structures as the average distance between two structures in the ensemble (a similar definition is used by **[16]** for weighted RMSD). Again, we assume all structures in the set are mutually compatible.

Definition 4.4. *We define function* $\mathrm{RMSD} : 2^{\mathcal{S}} \to \mathbb{R}$ *for a set of $m = |S|$ structures as*

$$\mathrm{RMSD}(S) = \sqrt{\begin{pmatrix} m \\ 2 \end{pmatrix}^{-1} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{RMSD}(S_i, S_j)^2}.$$

In order to find the best pairing of atoms in two structures we will need to reorder atoms in one of the structures. Formally, we define this using *automorphisms* of the structures.

Definition 4.5. *An automorphism of a structure $S = (\mathrm{Ids}, \mathrm{Sym}, \mathrm{Sel}, \mathrm{RIds}, \mathrm{Res}, \beta, \tau, \rho, \pi)$ is a bijective mapping $f : \mathrm{Sel} \to \mathrm{Sel}$ for which $\beta(x) = \beta(f(x))$, where $x \in \mathrm{Sel}$.*
We denote the set of all automorphisms of structure S by $\mathrm{Aut}(S)$.

Grouping of residues and atoms together will help to dramatically reduce the search space.

Definition 4.6. *A grouping g_A of a (finite) set A with respect to a key $k_A : A \to K$ (K is the set of keys) is a partitioning $A/\sim_{k_A}$, where $a \sim_{k_A} b \Leftrightarrow k_A(a) = k_A(b)$.*

Example 4.2. *The grouping of set $A = \{x, y, z, v, w\}$ with respect to key $k_A(x) = 1, k_A(y) = 1, k_A(z) = 2, k_A(v) = 1, k_A(w) = 2$ is a set $\{\{x, y, v\}, \{z, w\}\}$.*

Definition 4.7. *We say that groupings g_A and g_B are equivalent iff $|g_A| = |g_B|$ and $k_A((g_A)_i) = k_B((g_B)_i)$ for all suitable i, where k_A (k_B) is the key.*

Example 4.3. *Let $A = \{x, y, z, v, w\}$, $k_A(x) = 1, k_A(y) = 1, k_A(z) = 2, k_A(v) = 1, k_A(w) = 2$, $B = \{a, b, c, d, e\}$, and $k_B(a) = 1, k_B(b) = 1, k_B(c) = 2, k_B(d) = 1, k_B(e) = 2$. Then $g_A = \{\{x, y, v\}, \{z, w\}\}$, $g_B = \{\{a, b, d\}, \{c, e\}\}$, and $g_A \equiv g_B$.*

Theorem 4.1. *Let* $S = (\mathrm{Ids}, \mathrm{Sym}, \mathrm{Sel}, \mathrm{RIds}, \mathrm{Res}, \beta, \tau, \rho, \pi)$ *be a structure. Then size of* $\mathrm{Aut}(S)$ *is*

$$|\mathrm{Aut}(S)| = \prod_{i=1}^{|g|} |g_i|!,$$

where g *is the grouping of* Sel *with respect to* β*.*

PROOF. The proof is straightforward. □

Definition 4.8. *Paring of sets* A *and* B *(*$|A| = |B|$*) is a bijective mapping* $p : A \to B$*.*

Formally, we define the transformation of a structure by changing its coordinate function π.

Definition 4.9. *A transformation of a structure* $S = (\mathrm{Ids}, \mathrm{Sym}, \mathrm{Sel}, \mathrm{RIds}, \mathrm{Res}, \beta, \tau, \rho, \pi)$ *by function* $T : \mathbb{R}^3 \to \mathbb{R}^3$ *is a structure* $S' = (\mathrm{Ids}, \mathrm{Sym}, \mathrm{Sel}, \mathrm{RIds}, \mathrm{Res}, \beta, \tau, \rho, \pi')$ *where*

$$\pi'(x) = T(\pi(x)),$$

where $x \in \mathrm{Sel}$*.*

We define the average structure (or *model*) as the arithmetic average of positions of corresponding atoms of structures in the ensemble.

Definition 4.10. *An average structure* $\overline{S}$ *of non-empty set of* n *structures* $S \in 2^{\mathcal{S}}$*, where* $S_i = (\mathrm{Ids}_i, \mathrm{Sym}_i, \mathrm{Sel}_i, \mathrm{RIds}_i, \mathrm{Res}_i, \beta_i, \tau_i, \rho_i, \pi_i)$*, for all* i, j, k*:* $|\mathrm{Sel}_i| = |\mathrm{Sel}_j|$ *and* $\beta_i((\mathrm{Sel}_i)_k) = \beta_j((\mathrm{Sel}_j)_k)$*, is a structures* $\overline{S} = (\overline{\mathrm{Ids}}, \mathrm{Sym}, \overline{\mathrm{Sel}}, \overline{\mathrm{Res}}, \mathrm{RIds}_1, \mathrm{Res}_1, \beta_1, \tau_1, \rho_q, \overline{\pi})$ *where* $\overline{\mathrm{Ids}} = \overline{\mathrm{Sel}} = \mathrm{Sel}_1$*,* $\overline{\mathrm{Res}} = \mathrm{Res}_1$ *and*

$$\overline{\pi}(\overline{\mathrm{Ids}_i}) = \frac{1}{n} \sum_{j=1}^{n} (\mathrm{Sel}_j)_i.$$

Throughout the text, the following functions will be used to (implicitly) refer to the result of the corresponding algorithm. A detailed description of the algorithms behind the functions will be described in the individual sections of the text.

Definition 4.11. *We define a function* $\mathrm{superimpose} : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ *that takes two structures and returns the first structure superimposed onto the second, minimizing the value of RMSD. Implicit pairing of atoms is considered.*

Definition 4.12. *We define a function* $\text{best-fit} : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ *that takes two structures and returns the first structure superimposed onto the second, minimizing the value of RMSD. Optimal pairing of atoms is computed.*

Definition 4.13. *We define a function* $\mathrm{superimpose} : 2^{\mathcal{S}} \to 2^{\mathcal{S}}$ *that takes a set of structures and returns a new set where the structures are superimposed, minimizing the value of RMSD.*

Definition 4.14. *We define a function* $\mathrm{RMSD}_{\text{best-fit}}$ *as the value of RMSD after calling* best-fit*.*

CHAPTER 5

Superimposing Two Structures

Given two structures, the selection of atoms and their pairing (here we assume the pairing is given by good ordering of atoms in both structures), we must find a transformation that optimally superimposes them. It turns out there is a very elegant solution for this problem utilizing quaternion math, first published in [**14**] and more recently in [**15**]. This topic was also addressed in the author's bachelor thesis [**6**]. Quaternion math described in chapter 3 is used to solve a part of the problem.

This section describes a modified version of the algorithm presented in [**6**] based on the work of Coutias et al. in [**15**]. The running time (and space) complexity of the algorithm is linear in the number of selected atoms.

In the following, let $A, B \in \mathcal{S}$ and

$$A = (\mathrm{Ids}_A, \mathrm{Sym}_A, \mathrm{Sel}_A, \mathrm{RIds}_A, \mathrm{Res}_A, \beta_A, \tau_A, \rho_A, \pi_A)$$

and

$$B = (\mathrm{Ids}_B, \mathrm{Sym}_B, \mathrm{Sel}_B, \mathrm{RIds}_B, \mathrm{Res}_B, \beta_B, \tau_B, \rho_B, \pi_B).$$

Also, we assume that both structures are compatible as defined in 4.2.

The value of RMSD (defined in 4.3) serves as a measure of similarity of structures:

$$\mathrm{RMSD}(A, B) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|A_i - B_i\|^2}.$$

So, our goal is to find a transformation $T : \mathbb{R}^3 \to \mathbb{R}^3$ so that $\mathrm{RMSD}(T(A), B)$ is minimized:

$$T = \arg\min_{\phi} \mathrm{RMSD}(\phi(A), B). \tag{5.1}$$

The transformation T must not deform the structure in any way. Therefore, we consider only rotations and translations. Later, improper rotations (i.e. a reflection followed by a rotation) are taken into account as well. Decomposing T into a rotation R and a translation vector t yields

$$\mathrm{RMSD}(T(A), B) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|R(A_i) + t - B_i\|^2}. \tag{5.2}$$

In the following text, firstly, a method for finding the optimal translation is described. Then, a quaternion based method for finding the optimal rotation. Next, improper rotations are considered which requires a minor modification of the algorithm. Afterwards, the uniqueness of the solution is discussed. Finally, we present a pseudo-code of the basic algorithm and its running time is analyzed.

1. Translation

It turns out that the optimal alignment superimposes geometrical centers of both structures [**14**].

Definition 5.1. *We define a function* $\mathrm{center} : \mathcal{S} \to \mathbb{R}^3$ *as*

$$\mathrm{center}(S) = \frac{1}{n}\sum_{i=1}^{n} S_i.$$

Now, we will express coordinates of both structures with respect to their geometric centers (i.e. the center being the origin of the coordinate system) by changing their respective coordinate functions π to get

$$\pi'_A(x) = \pi_A(x) - \text{center}(\pi_A(C_A)), \tag{5.3}$$

and

$$\pi'_B(x) = \pi_B(x) - \text{center}(\pi_B(C_B)). \tag{5.4}$$

Lemma 5.1. *Let $A \in \mathcal{S}$ be a structure with n (selected) atoms and $\text{center}(A) = 0$. Then*

$$\sum_{i=1}^{n} A_i = 0.$$

PROOF. Straightforward from definition of center. □

Theorem 5.1. *Let $A, B \in \mathcal{S}$ be structures with coordinates expressed with respect to their geometric center (i.e. $\text{center}(A) = \text{center}(B) = 0$). Then the translation part t of transformation T minimizing $\text{RMSD}(T(A), B)$ is $t = 0$.*

PROOF. Using the identity $\|a\|^2 = a \cdot a$ and the distributivity of dot product the expression

$$\sum_{i=1}^{N} \|T(A_i) - B_i\|^2 = \sum_{i=1}^{n} \|R(A_i) + t - B_i\|^2$$

can be expanded to obtain

$$\sum_{i=1}^{n} \|R(A_i) - B_i\|^2 + 2t \cdot \sum_{i=1}^{n} (R(A_i) - B_i) + \sum_{i=1}^{n} \|t\|^2.$$

From the lemma 5.1, the second term $2t \cdot \sum_{i=1}^{n}(R(A_i) - B_i)$ is equal to 0. We are now left with

$$\sum_{i=1}^{n} \|R(A_i) - B_i\|^2 + \sum_{i=1}^{n} \|t\|^2.$$

This expression is obviously minimized iff $t = 0$, since $\|\cdot\|^2$ is a non-negative function. □

This leaves us with the task to find a rotation R that minimizes

$$\sum_{i=1}^{n} \|R(A_i) - B_i\|^2. \tag{5.5}$$

2. Rotation

In order to successfully utilize quaternions for finding the optimal rotation, we must first reformulate our problem somewhat.

From now on we assume that structures A and B have the coordinates of their atoms expressed with respect to their geometric centers (i.e. $\text{center}(A) = \text{center}(B) = 0$). Using the properties of the vector norm and dot product we expand $\sum_{i=1}^{n} \|R(A_i) - B_i\|^2$ to obtain

$$\sum_{i=1}^{n} \|R(A_i) - B_i\|^2 = \sum_{i=1}^{n} \|R(A_i)\|^2 + \sum_{i=1}^{n} \|B_i\|^2 - 2\sum_{i=1}^{n} R(A_i) \cdot B_i. \tag{5.6}$$

Because rotation preserves vector norm, the first two terms are constant. Therefore, in order to minimize the value of the above expression we need to find a rotation R that maximizes the value of

$$\sum_{i=1}^{n} R(A_i) \cdot B_i. \tag{5.7}$$

This is where quaternions come in. Using the definition 3.14 we can write rotation R as

$$R(x) = qxq^*, \tag{5.8}$$

where q is a suitable unit quaternion and vector x is expressed as a purely imaginary quaternion. We obtain

$$\sum_{i=1}^{n} R(A_i) \cdot B_i = \sum_{i=1}^{n} qA_iq^* \cdot B_i. \tag{5.9}$$

Using identities from lemmas 3.2, 3.3 and 3.4 the previous expression can be rewritten as

$$\sum_{i=1}^{n} (qA_i) \cdot (B_iq). \tag{5.10}$$

The next step is to express the left and right quaternion multiplication from the previous expression using matrix operations as

$$qA_i = \hat{\mathbf{A}}_i q = \begin{pmatrix} 0 & -a_{i,x} & -a_{i,y} & -a_{i,z} \\ a_{i,x} & 0 & a_{i,z} & -a_{i,y} \\ a_{i,y} & -a_{i,z} & 0 & a_{i,x} \\ a_{i,z} & a_{i,y} & -a_{i,x} & 0 \end{pmatrix} q, B_iq = \mathbf{B}_i q = \begin{pmatrix} 0 & -b_{i,x} & -b_{i,y} & -b_{i,z} \\ b_{i,x} & 0 & -b_{i,z} & b_{i,y} \\ b_{i,y} & b_{i,z} & 0 & -b_{i,x} \\ b_{i,z} & -b_{i,y} & b_{i,x} & 0 \end{pmatrix} q$$

to obtain

$$\sum_{i=1}^{n} (\hat{\mathbf{A}}_i q) \cdot (\mathbf{B}_i q), \tag{5.11}$$

which, can further be simplified to

$$\sum_{i=1}^{n} q^T \hat{\mathbf{A}}_i^T \mathbf{B}_i q. \tag{5.12}$$

Factoring out q and q^T we obtain

$$q^T \left(\sum_{i=1}^{n} \hat{\mathbf{A}}_i^T \mathbf{B}_i \right) q. \tag{5.13}$$

By letting $\mathbf{N}_i = \hat{\mathbf{A}}_i^T \mathbf{B}_i$ and $\mathbf{N} = \sum_{i=1}^{n} \mathbf{N}_i$ we obtain

$$q^T \mathbf{N} q. \tag{5.14}$$

We are looking for a quaternion (4D vector) q that maximizes the above expressions. It turns out that q is the (right) eigenvector of $\mathbf{N}$ corresponding to its maximum positive eigenvalue.

Before we can prove the statement of the previous paragraph we first need to show several properties of $\mathbf{N}$.

Lemma 5.2. $\mathbf{N}$ *is a symmetric traceless matrix.*

PROOF. Indeed, each term in the sum $\sum_{i=1}^{n} \mathbf{N}_i$ has the form

$$\begin{pmatrix} a_x b_x + a_y b_y + a_z b_z & a_y b_z - a_z b_y & a_z b_x - a_x b_z & a_x b_y - a_y b_x \\ a_y b_z - a_z b_y & a_x b_x - a_y b_y - a_z b_z & a_y b_x + a_x b_y & a_z b_x + a_x b_z \\ a_z b_x - a_x b_z & a_y b_x + a_x b_y & -a_x b_x + a_y b_y - a_z b_z & a_z b_y + a_y b_z \\ a_x b_y - a_y b_x & a_z b_x + a_x b_z & a_z b_y + a_y b_z & -a_x b_x - a_y b_y + a_z b_z \end{pmatrix}$$

which is obviously a symmetric matrix. Therefore, $\mathbf{N} = \sum_{i=1}^{n} \mathbf{N}_i$ is also a symmetric matrix. Finally, we have $\mathrm{tr}(\mathbf{N}_i) = 0$ for each $\mathbf{N}_i$, therefore $\mathrm{tr}(\mathbf{N}) = 0$. □

Theorem 5.2. *Matrix* $\mathbf{N} \neq \mathbf{0}$ *contains at least one positive eigenvalue.*

PROOF. From lemma 5.2, $\mathbf{N}$ is a symmetric traceless matrix. From Spectral Decomposition Theorem **[17]** we can express $\mathbf{N}$ in the form $\mathbf{N} = \mathbf{PDP}^{-1}$ (i.e. $\mathbf{N}$ can be expressed in diagonal form). Finally, $\mathrm{tr}(\mathbf{PDP}^{-1}) = \mathrm{tr}(\mathbf{D})$ **[18]**.

It follows that

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 0,$$

where λ_i is the i-th eigenvalue of $\mathbf{N}$.

Therefore, there must exist at least one i so that $\lambda_i > 0$. □

We are now ready to prove the final theorem of this chapter.

Theorem 5.3. *Proper rotation* R *maximizing* $\sum_{i=1}^{n} R(A_i) \cdot B_i$ *is represented by a quaternion* $q = v_{max}$*, where* v_{max} *is the (right) eigenvector of* $\mathbf{N}$ *corresponding to the maximum positive eigenvalue of* $\mathbf{N}$*.*

PROOF. Since $\mathbf{N}$ is a symmetric matrix, its normalized eigenvectors v_i ($i = 1, 2, 3, 4$) form an orthonormal basis of $\mathbb{R}^4$ (from Spectral Decomposition Theorem **[17]**). Expressing q as a vector in basis v_i we have

$$q = \sum_{i=1}^{4} \alpha_i v_i$$

with $\sum_{i=1}^{4} \alpha_i^2 = 1$ which follows from q being a unit quaternion.

The product $\mathbf{N}q$ can then be written as

$$\mathbf{N}q = \sum_{i=1}^{4} \alpha_i \lambda_i v_i,$$

where λ_i is the eigenvalue corresponding to the eigenvector v_i. Then, we can rewrite (5.14) to

$$q^T \mathbf{N} q = \sum_{i=1}^{4} \alpha_i \lambda_i q^T v_i = \sum_{i=1}^{4} \alpha_i \lambda_i \left(\sum_{j=1}^{4} \alpha_j v_j^T \right) v_i.$$

Since v_i form an orthonormal basis of $\mathbb{R}^4$, $v_i^T v_j = \delta_{ij}$, where δ_{ij} is Dirac delta function (i.e. if $i = j$ return 1, otherwise return 0). The previous expression now simplifies to

$$\sum_{i=1}^{4} \alpha_i \lambda_i \alpha_i v_i^T v_i = \sum_{i=1}^{4} \alpha_i^2 \lambda_i,$$

which is a weighted mean ($\sum_{i=1}^{4} \alpha_i^2 = 1$ and $\alpha_i^2 \geq 0$). A weighted mean has a maximum when the biggest number $\lambda_{\max}$ is weighted with $\alpha_{\max} = 1$ and all other α_i are zero. This corresponds to choosing the rotation quaternion q as

$$q = v_{\max},$$

where $v_{\max}$ is the (normalized) eigenvector corresponding to the maximum positive eigenvalue $\lambda_{\max}$ (which exists as shown in lemma 5.2). □

3. Improper Rotations

Sometimes, an improper rotation (reflection followed by a rotation) might yield a better result [**15**].

Incorporating improper rotations can be handled by considering the maximum *absolute* eigenvalue as well, its corresponding eigenvector, and the inversion transformation $f(x) = -x$. Together, this transformation can be elegantly represented by a rotation (transformation) matrix

$$\mathbf{R} = \operatorname{sign}(\lambda_{\max})\mathbf{Q}, \tag{5.15}$$

where $\mathbf{Q}$ is the rotation matrix corresponding to quaternion $v_{\max}$ (see theorem 3.1) and $v_{\max}$ is the eigenvector of $\mathbf{N}$ corresponding to its largest *absolute* eigenvalue.

We consider the improper rotation separately, because they are not always allowed. This is because for a *chiral* structure, the inversion operation encompasses the change-oh-hand of the coordinate system of the structure as well. In a physical system, this aspect if not desirable.

4. Computing the Value of RMSD

Finally, we can compute the value of RMSD without transforming the first structure using the eigenvalue $\lambda_{\max}$.

Theorem 5.4. *Let $A, B \in \mathcal{S}$. Then*

$$\operatorname{RMSD}(R(A), B) = \sqrt{\frac{1}{n}\left(\sum_{i=1}^{n}\|A_i\|^2 + \sum_{i=1}^{n}\|B_i\|^2 - 2|\lambda_{max}|\right)},$$

where λ_{max} is the maximum eigenvalue from the proof 5.3.

PROOF. This follows from equation 5.6, rotations preserving the vector norm, and the fact that $q^T\mathbf{N}q = \lambda_{\max}$ (follows from theorem 5.3). Finally, $\lambda_{\max}$ is in absolute value, because, in general, we consider improper rotations as well. □

5. Uniqueness of the Solution

From the algorithm itself it is straightforward to deduce that the solution is unique provided that the individual eigenvalues λ_i are all different.

Nevertheless, there are two basic degenerate cases when the solution is not unique:

(1) One structure lies on a line (the points of the structure are linearly dependent and there are at least two non-equal points) and the other does not (the points of the structure are *not* linearly dependent).

    In this case the solution is a family of all rotations around the axis defined by the first structure.

(2) All points of the first structure lie in a single point and the second structure contains at least two non-equal points.

    In this case any rotation is a possible solution.

Fortunately, the structures we are interested in practically never exhibit the mentioned degeneracy. There are examples of molecules that have the type 1 degeneracy, such as CO_2, but these are very rare and not relevant to the topic of this thesis (binding sites in proteins). Of course, the type 2 degeneracy can never occur in a real molecule.

6. Algorithm

The following text contains a short summary of steps needed to superimpose two structures:

(1) Express structures A and B in coordinates with respect to their geometrical center (see section 1).

(2) Compute the matrix $\mathbf{N}$ (see equation 5.14).

(3) Find the largest positive eigenvalue $\lambda_{\max}$ of $\mathbf{N}$. If improper rotations are allowed, consider the largest absolute eigenvalue.
(4) Find the eigenvector $v_{\max}$ of $\mathbf{N}$ corresponding to eigenvalue $\lambda_{\max}$.
(5) Return the transformation $\mathbf{R} = \operatorname{sign}(\lambda_{\max})\mathbf{Q}$ and the value of RMSD (see section 3).

6.1. Pseudo-code. Pseudo-code 5.1 provides a more detailed description of the algorithm.

Algorithm 5.1 Function $\operatorname{superimpose}(A, B)$.

Input: Structures A and B with their coordinates with geometric center as origin.
Output: A 3x3 matrix representing thr transformation of the structure A and the value of RMSD.

$\mathbf{N} \leftarrow \sum_{i=1}^{n} \operatorname{quaternionRightMultiplicationMatrix}(A_i)^T \operatorname{quaternionLeftMultiplicationMatrix}(B_i)$
if improper rotation is allowed **then**
    $\lambda_{\max} \leftarrow$ eigenvalue of $\mathbf{N}$ with the maximum *absolute* value
else
    $\lambda_{\max} \leftarrow$ eigenvalue of $\mathbf{N}$ with the maximum value
end if
$\text{rmsd} \leftarrow \sqrt{\frac{1}{n}\left(\sum_{i=1}^{n} \|A_i\|^2 + \sum_{i=1}^{n} \|B_i\|^2 - 2|\lambda_{\max}|\right)}$
$q \leftarrow \operatorname{eigenvector}(\lambda_{\max})$
if $\lambda_{\max} < 0$ **then**
    rotation $\leftarrow -\operatorname{rotationMatrix}(q)$
else
    rotation $\leftarrow \operatorname{rotationMatrix}(q)$
end if
return $(\text{rotation}, \text{rmsd})$

7. Running Time and Space Complexity

The running time of the algorithm is linear in the number of atom in the structure. Finding the eigenvalues of the eigenvectors of a 4x4 matrix can be computed in constant time. All other steps are linear.

Similarly, the space complexity is also linear in the number of atoms.

CHAPTER 6

Finding the Best Pairing of Atoms

In the previous chapter we presented an algorithm for computing the best alignment of a pair of structures given the fact that we know the atom pairing a priori.

In this chapter an algorithm is presented that finds the optimal pairing of atoms of participating structures. By pairing we mean ordering the atoms of both structures so that i-th atoms correspond to each other. This can be achieved by scanning through automorphisms of one of the structures (i.e. reordering the atoms of one of the structures) and then applying the algorithm for pair-wise superimposition from the previous chapter.

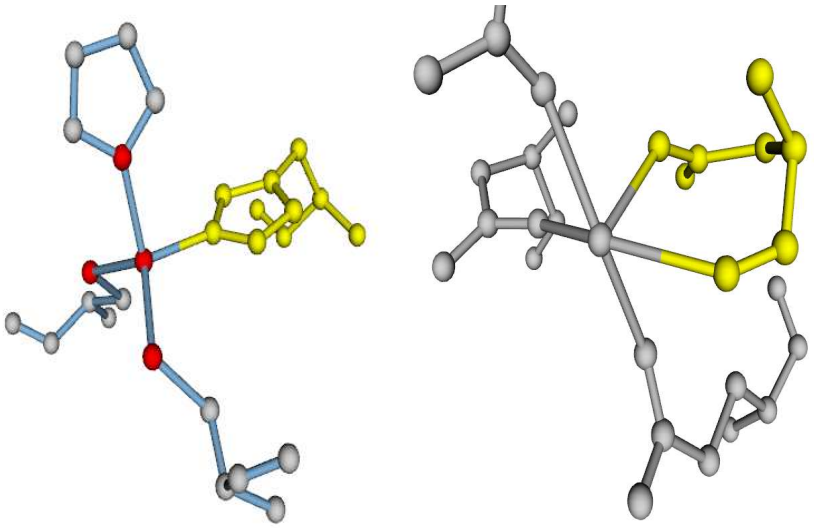
The mathematical formulation of the problem is to find a function f for which

$$f = \underset{\phi \in \mathrm{Aut}(A)}{\arg\min}\, \mathrm{RMSD}(\mathrm{superimpose}(\phi(A), B), B). \tag{6.1}$$

The complexity of finding function f is obviously tied to the size of $\mathrm{Aut}(A)$. As we have seen in theorem 4.1, the size of $\mathrm{Aut}(A)$ is exponential in number of atoms in A (or selection of A, to be more precise). This is of course very undesirable. Fortunately, we can do better by utilizing information about the structure of proteins (or, in fact, any other structure). In theorem 4.1, the atoms were grouped based on their element symbol. In this chapter, we will show that grouping the atoms based on the information about their residues will often result in a dramatic reduction of the search space. The resulting complexity will still remain exponential in nature, but the exponent will get much lower, allowing us to superimpose larger structures in reasonable time. Also, this will come at the price of sometimes not finding the best possible pairing because a heuristic will be used to pair atoms on corresponding residues.

1. Types of Binding Sites

There are two basic types of protein binding sites:



(a) ZnS_2N_2 (b) CaO_5

FIGURE 6.1. Types of binding sites.

(1) The central atom (i.e. the metal such as Zn) binds to elements that are each on a residue with a different identifier. An example is the ZnN_2S_2 binding site shown on figure 6.1a.
(2) The central atom binds to elements, at least two of which being on a the same residue. An example is the CaO_5 binding site shown on figure 6.1b (the residue of interest is highlighted in yellow color).

Based on the type of the binding site and the subset of atoms we want to fit the structures by, the search space for finding the minimum from equation 6.1 can be reduced.

However, let us first look at several examples to better illustrate the problem:

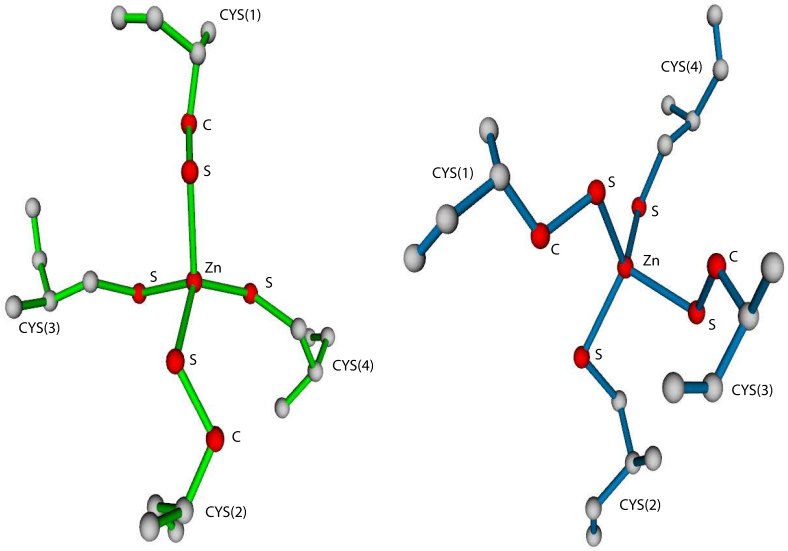


FIGURE 6.2. Two ZnS_4 structures.

Example 6.1. *Figure 6.2 shows two ZnS_4 binding sites of type 1 and $\{Zn, S, S, S, S, C, C\}$ subset (atoms marked by red).*

Both structures contain five residues: 4x CYS (1-4) and 1x ZN.

The selection breakdown based on the residue identifier and the residue name on the first structure (green) is

ZN: *Zn*
CYS(1): *S, C*
CYS(2): *S, C*
CYS(3): *S*
CYS(4): *S*

On the second structure (blue) we have

ZN: *Zn*
CYS(1): *S, C*
CYS(2): *S*
CYS(3): *S, C*
CYS(4): *S*

The question is how many automorphisms of the first structure do we have to test in order to find the optimal superimposition (i.e. minimize RMSD). The answer is 4.

Obviously, we only have to test mappings where atoms on $\mathrm{CYS_A}$ are mapped to atoms on $\mathrm{CYS_B}$ containing the same number of atoms. Moreover, since each residue contains a maximum of one atom with a given element symbol, the pairing on each residue is unique. Therefore, the number of pairing we have to test is $1!2!2! = 4$.

Testing all possible pairing of atoms based on their element symbol we would have to test $1!2!4! = 48$ cases.

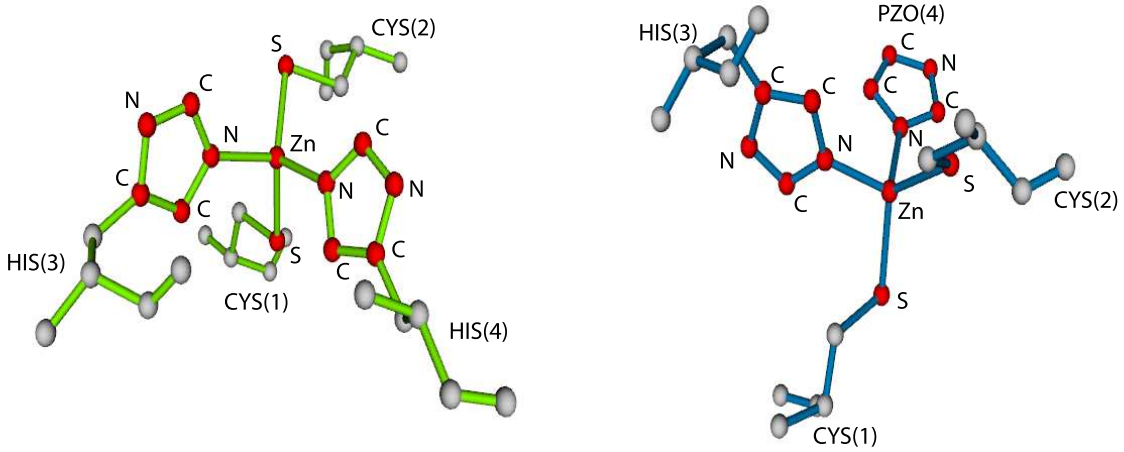


FIGURE 6.3. Two ZnN_2S_2 structures.

Example 6.2. *Figure 6.3 shows two ZnN_2S_2 binding sites of type 1 and $\{Zn, C, C, C, C, C, C, N, N, N, N, S, S\}$ subset (atoms marked by red).*

The green structure contains 2x CYS and 2x HIS residues and the blue has 2x CYS, 1x HIS and 1x PZO.

The selection breakdown based on the residue identifier and the residue name on the first structure (green) is

ZN: *Zn*
CYS(1): *S*
CYS(2): *S*
HIS(3): *C, C, C, N, N*
HIS(4): *C, C, C, N, N*

And the selection breakdown on the second structure (blue) is

ZN: *Zn*
CYS(1): *S*
CYS(2): *S*
HIS(3): *C, C, C, N, N*
PZO(4): *C, C, C, N, N*

In this case, we will want to (possibly) match HIS residue to PZO, because the selections on both HIS and PZO form a cycle containing three carbon and two nitrogen atoms.

Similarly to the previous example, the number of ways we can match residues is $2!2! = 4$. Nevertheless, this time the correspondence of atoms on HIS-HIS and HIS-PZO pairs is not unique. Fortunately, knowing that the atoms lie on the same residue, we can match carbon and nitrogen atoms independently, resulting in $3! + 2! = 10$ complexity for each pair. There are two pairs, therefore, we must multiply this number by 2. Together, the number of automorphisms we have to test is $1!2!\,[2(3! + 2!)]\,2! = 80$.

Now, testing all possible pairing of atoms based on their element symbol, the complexity would be $1!2!6!4! = 34560$. It might be tempting to try and match the atoms independently as in the previous paragraph (which would yield $2! + 6! + 4! = 746$ pairs), but that is not possible. The reason is that atoms from different residues could get mixed up. For example, the C atoms from $\mathrm{HIS}(3)_\mathrm{A}$ might end matched to C atoms on $\mathrm{PZO}(4)_\mathrm{B}$ while the N atoms from $\mathrm{HIS}(3)_\mathrm{A}$ would end up being matched to N atoms on $\mathrm{HIS}(3)_B$.

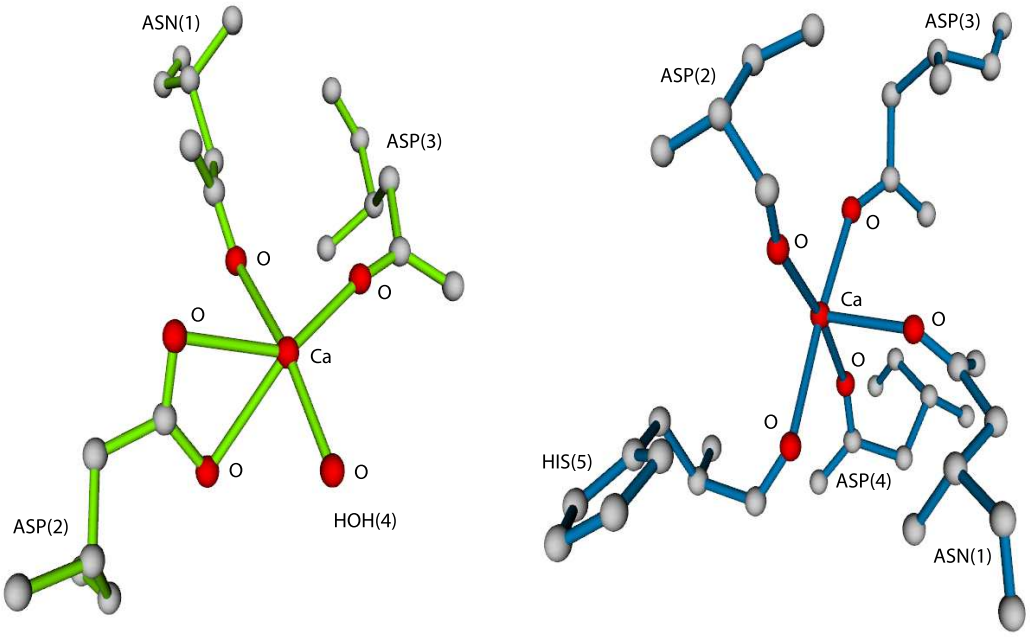


FIGURE 6.4. Two CaO_5 structures.

Example 6.3. *Figure 6.4 shows CaO_5 of type 2 and CaO_5 of type 1 binding sites with $\{Ca, O, O, O, O, O\}$ selection (atoms marked by red).*

The selection breakdown based on the residue identifier and the residue name on the first structure (green) is

CA: *Ca*
ASN(1): *O*
ASP(2): *O,O*
ASP(3): *O*
HOH(4): *O*

And the selection breakdown on the second structure (blue) is

CA: *Ca*
ASN(1): *O*
ASP(2): *O*
ASP(3): *O*
ASP(4): *O*
HIS(5): *O*

This time around, each structure contains different number of residues. In this case, only the information about element symbol is helpful. Therefore, we have to check all possible pairings and the overall complexity is $1!5! = 120$. Fortunately, in praxis this case is rather rare.

We will finish the example section by one negative example.

Example 6.4. *A reader familiar with the PDB format might notice that naming of atoms is not included in our definition of* structure*, whereas, at the first sight, it might still provide useful information about the structure of the binding site. Unfortunately, this is not so.*

Ideally, it would be possible to pair the atoms of both structures based on their name as the structure of amino acid is well-defined (in PDB format, each atom on each amino acid has a unique name). Nevertheless, in the general case, and particularly when dealing with binding sites, this is not possible.

One example are two binding sites of the type ZnN_2S_2 shown in figure 6.5. The first structure has both HIS residues bound at the nitrogen atoms named ND1. Nevertheless, the other structure has the first HIS residue bound at the atom named ND1 and the second at NE1. Now, suppose we were to fit these two structures using the $\{Zn, S, S, N, N, N, N\}$ selection. On both structures, ND1 and NE1

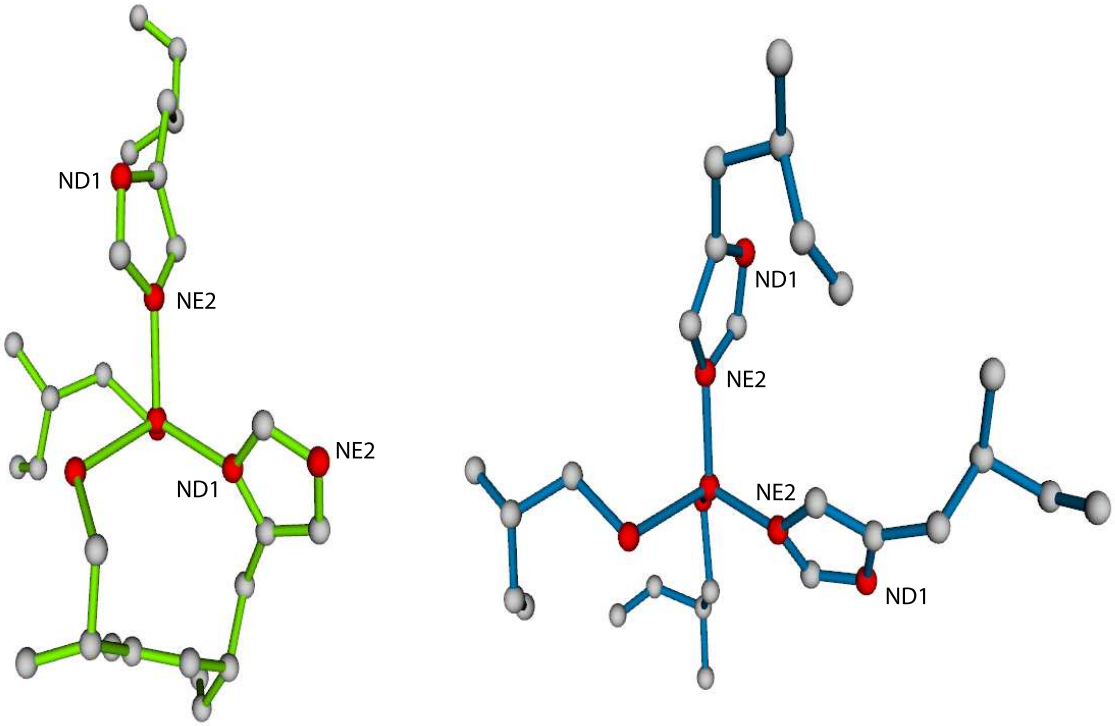


FIGURE 6.5. Two ZnN_2S_2 structures.

atoms would be selected but pairing them based on their name would yield an incorrect (not optimal) result – the ND1 atom on the green structure that is adjacent to the Zn atom would be paired to ND1 on the blue structure that is not adjacent to the Zn atom.

We might, of course, consider an even simpler case: the subset $\{Zn, S, S, N, N\}$*. In this case no pairing based on atom names would be possible. Nevertheless, it is still meaningful to pair ND1 to NE1 when computing a model of the binding site.*

2. Grouping

As hinted by the first three examples in the previous section, we will use three different groupings to reduce the search space. These groupings are called *residue name*, *residue*, and *element symbol*.

In the first two cases, we will use two level grouping (in the sense of definition 4.6). We call these groups *top-level* group and *bottom-level* group. The last case does not necessarily need two level grouping, however, for consistency reasons it will be represented as such as well.

First, the *bottom-level* grouping will be formed by grouping atoms with the same residue identifier and residue name, and then grouping each group based on the atoms' element symbols.

Then, the *top-level* grouping will consist of grouping some elements of the *bottom-level* grouping together.

Example 6.5. *Let us consider the set* $\{1, 2, 3, 4, 5, 6, 8, 9\}$.

A bottom-level grouping (the key is arbitrary) of this set might look like

$$\{\{\{1, 2\}, \{3\}\}, \{\{4, 5\}, \{6\}\}, \{\{8, 9\}\}\}.$$

Then, in the top-level grouping (again, the key is arbitrary), $\{\{1, 2\}, \{3\}\}$ *and* $\{\{4, 5\}, \{6\}\}$ *are grouped together and* $\{\{8, 9\}\}$ *left alone, resulting in*

$$\{\{\{\{1, 2\}, \{3\}\}, \{\{4, 5\}, \{6\}\}\}, \{\{\{8, 9\}\}\}\}.$$

In the following, let

$$A = (\mathrm{Ids}_A, \mathrm{Sym}_A, \mathrm{Sel}_A, \mathrm{RIds}_A, \mathrm{Res}_A, \beta_A, \tau_A, \rho_A, \pi_A)$$

and

$$B = (\mathrm{Ids}_B, \mathrm{Sym}_B, \mathrm{Sel}_B, \mathrm{RIds}_B, \mathrm{Res}_B, \beta_B, \tau_B, \rho_B, \pi_B).$$

We assume that structures A and B are compatible in the sense of definition 4.2.
We consider these groupings:

Residue name: The *bottom-level* grouping is grouped with a key based on residue identifier and residue name, and then grouped by element symbol. Formally, the *bottom-level* grouping $g_{A_{\mathrm{bottom}}}$ for structure A, $x \in \mathrm{Sel}_A$, has the key

$$k_{A_{\mathrm{bottom}}}(x) = (\tau_A(x), \rho_A(\tau_A(x))).$$

Then, each subgroup in $g_{A_{\mathrm{bottom}}}$ is grouped using the key $k(x) = \beta_A(x)$ (the element symbol of x).

Next, the *top-level* grouping is established with a key based on residue name and atoms selected on the given residue. Formally, grouping $g_{A_{\mathrm{top}}}$ has the key

$$k_{A_{\mathrm{top}}}(x) = \rho'_A(x),$$

where ρ'_A is the natural extension of ρ_A to grouping of elements (each top level group contains elements with the same residue name).

Similarly for structure B.

Example 6.6. *An example of residue name grouping is shown.*

Considering the first structure from example 6.2 with the selected atoms $\{Zn, C, C, C, C, C, C, N, N, N, N, S, S\}$ *and the selection breakdown based on the residue identifier and the residue name:*

ZN: *Zn*
CYS(1): *S*
CYS(2): *S*
HIS(3): *C, C, C, N, N*
HIS(4): *C, C, C, N, N*

the first phase of bottom-level *grouping is*

$$\{\{Zn\}, \{C, C, C, N, N\}, \{C, C, C, N, N\}, \{S\}, \{S\}\}.$$

Next, the atoms in each subgroup are grouped by element symbol resulting in

$$\{\{\{Zn\}\}, \{\{C, C, C\}, \{N, N\}\}, \{\{C, C, C\}, \{N, N\}\}, \{\{S\}\}, \{\{S\}\}\}.$$

Finally, the top-level *grouping yields*

$$\{\{\{\{Zn\}\}\}, \{\{\{C, C, C\}, \{N, N\}\}, \{\{C, C, C\}, \{N, N\}\}\}, \{\{\{S\}\}, \{\{S\}\}\}\}.$$

Residue: The *bottom-level* grouping $g_{A_{\mathrm{bottom}}}$ is the same as in the previous case, where the key was based on on residue identifier and residue name:

$$k_{A_{\mathrm{bottom}}}(x) = (\tau_A(x), \rho_A(\tau_A(x))).$$

Again, each subgroup in $g_{A_{\mathrm{bottom}}}$ is grouped using the key $k(x) = \beta_A(x)$ (the element symbol of x).

Next, the *top-level* $g_{A_{\mathrm{top}}}$ grouping is grouped with a key based on atoms selected on the given residue. Formally, the key is just an identity mapping:

$$k_{A_{\mathrm{top}}}(x) = x.$$

Similarly for structure B.

Element symbol: In this case, the *bottom-level* grouping is grouped with a key based on element identifier, which is nothing else than an identity mapping. Formally, the *bottom-level* grouping $g_{A_{\text{bottom}}}$ has the key

$$k_{A_{\text{bottom}}}(x) = x.$$

This means that each group will be a singleton containing one atom.

Now, for consistency reasons we have "group" each group in $g_{A_{\text{bottom}}}$ using the key $k(x) = \beta_A(x)$.

Finally, the *top-level* grouping $g_{A_{\text{top}}}$ is based on element symbols and has the key

$$k_{A_{\text{top}}}(x) = \beta'_A(x)$$

where β'_A is the natural extension of β_A to set elements.

Similarly for structure B.

Now, in order to establish the grouping of structures A and B, both (two-level) grouping g_A and g_B have to be *equivalent*.

Definition 6.1. *We say that (two-level) groupings g_A and g_B of structures $A, B \in \mathcal{S}$ are equivalent (denoted by $A \equiv B$) iff $|g_A| = |g_B|$ and $(g_A)_i \sim (g_B)_i$ for each suitable i, where $\sim$ is defined as $x \sim y \Leftrightarrow |x| = |y| \wedge \forall i \forall j.\ x_i$ and y_j are equivalent groupings (as in definition 4.7).*

What the previous definition says is that (two-level) groupings are equivalent if they contain the same number of groups and each subgroup contains mutually compatible sets (that contain atoms with the same element symbol).

A convenient way to establish a grouping is to first group both A and B using *residue name* grouping and then check if both groupings are equivalent. If not, test *residue* grouping. Finally, if no *residue* grouping exists, *element symbol* grouping is established (*element symbol* grouping always exists for compatible structures).

3. Matching of *Top-level* Groups

At this point we have equivalent groupings g_A and g_B of structures A and B.

We know that the i-th members of g_A and g_B correspond to each other, so our goal is to establish a pairing of subsets in each corresponding subgroup of g_A and g_B.

To achieve this, we scan the groups recursively. We start with the first group and continue:

- For each permutation of elements of the i-th group: match the corresponding subgroups and if the last group was not reached, proceed to the $(i+1)$-th group.
- If the last group was reached, call superimpose (described in the previous chapter) on the "current" pairing. If the result is better than the already found one, store it.
- When all permutations have been visited return the best found result.

Example 6.7. *Let*

$$g_A = \{\{\{\{S\}\}_1, \{\{S\}\}_2\}, \{\{\{Zn\}\}\}\}$$

and

$$g_B = \{\{\{\{S\}\}_3, \{\{S\}\}_4\}, \{\{\{Zn\}\}\}\}$$

be groupings of some (arbitrary) structures A and B (the bottom indexes are used to distinguish individual groups). Obviously, the pairings are equivalent in the above defined sense.

There are two top-level *groups: $\{\{\{S\}\}, \{\{S\}\}\}$ and $\{\{\{Zn\}\}\}$.*

The algorithm will work as following:

(1) *Match the first permutation $\{\{\{S\}\}_1, \{\{S\}\}_2\}$ of g_A to first group of $\{\{\{S\}\}_3, \{\{S\}\}_4\} \in g_B$. It means that $\{\{S\}\}_1$ is matched to $\{\{S\}\}_3$ and $\{\{S\}\}_2$ is matched to $\{\{S\}\}_4$. Pairing $S_1 - S_3, S_2 - S_4$ is produced.*

(2) *Last group was not yet matched, therefore match the second group: $\{\{\{Zn\}\}\} \in g_A$ to $\{\{\{Zn\}\}\} \in g_B$. Pairing $Zn - Zn$ is produced.*

(3) *Last group was matched. Call* superimpose *and store the result.*

(4) *Match the second permutation of* $\{\{\{S\}\}_2, \{\{S\}\}_1\}$ *of* g_A *to first group of* $\{\{\{S\}\}_3, \{\{S\}\}_4\} \in g_B$*. It means that* $\{\{S\}\}_1$ *is matched to* $\{\{S\}\}_4$ *and* $\{\{S\}\}_2$ *is matched to* $\{\{S\}\}_3$*. Pairing* $S_1 - S_4, S_2 - S_3$ *is produced.*
(5) *Last group was not yet matched, therefore match the second group:* $\{\{\{Zn\}\}\} \in g_A$ *to* $\{\{\{Zn\}\}\} \in g_B$*. Pairing* $Zn - Zn$ *is produced.*
(6) *Last group was matched. Call* superimpose *and store the result if it is better than the previously found one.*
(7) *All permutations have been checked, return the best found result.*

4. Matching of *Bottom-level* Groups

Unlike in the previous case, *bottom-level* groups are matched independently. Therefore, we can simply iterate through individual groups instead of using a recursive approach.

For each element S_A of the subgroup of the *bottom-level* group of first structure (which is the set of atoms with the same element symbol) and the corresponding subgroup S_B of the second structure perform the following:

- For each permutation of elements in S_A and the corresponding subset of S_B, call function superimpose (described in the previous chapter) and store the value of RMSD.
- Return the permutation (or implicit pairing induced by the permutation, to be more precise) with minimal RMSD.

Of course, if $|S_A| = 1$ then superimpose is not called, because the pairing is unique and is handled by the *top-level* matching described above.

Example 6.8. *Let*

$$X = \{\{C_1, C_2\}, \{N\}\}$$

and

$$Y = \{\{C_3, C_4\}, \{N\}\}$$

be bottom-level *groups of some (arbitrary) structures* A *and* B *(the bottom indexes are used to distinguish individual atoms).*

There are two subgroups in each group: $\{C, C\}$ *and* $\{Zn\}$*.*

The algorithm will work as following:

(1) *For the first permutation* $\{C_1, C_2\}$ *establish pairing* $C_1 - C_3, C_2 - C_4$*, call* superimpose*, and store the result.*
(2) *For the second permutation* $\{C_2, C_1\}$ *establish pairing* $C_1 - C_4, C_2 - C_3$*, call* superimpose*, and store the result if it is better than the previous step.*
(3) *Establish pairing* $N - N$*.*
(4) *Return the result.*

Unfortunately, matching each subgroup independently does not always produce the best result. In some cases, when the structure exhibits some degree of symmetry, several pairings might produce a "good" value of RMSD. Therefore, a locally better value of RMSD can be chosen, that produces a worse overall result.

There are several ways around this problem such as choosing *pivot* elements or considering the bonds between individual atoms. Nevertheless, these are left as the future work on the subject.

5. Size of the Grouping

The following theorem gives a formula for calculating the size of a grouping – the number of automorphisms that need to be tested in order to find the best pairing of two structures (in other words the number of calls to the function $\mathrm{superimpose}$ that need to be performed).

Theorem 6.1. *Let g be a two-level grouping with n* top-level *groups. Then the size of the grouping is*

$$\mathrm{size}(g) = \prod_{i=1}^{n} \left(|g_i|!\sigma(g_i)\right),$$

where

$$\sigma(x) = \begin{cases} 1, & \text{if } \sigma'(x) = 0; \\ \sigma'(x), & \text{otherwise.} \end{cases}$$

$$\sigma'(x) = |x| \sum_{j=1}^{|x|} \chi(x_j)$$

$$\chi(x) = \begin{cases} 0, & \text{if } x = 1; \\ x!, & \text{otherwise.} \end{cases}$$

PROOF. For each *top-level* group g_i there are two options:

(1) The correspondence of atoms in g_i is unique. In this case, the subgroup does not "generate" any extra automorphisms we have to check.
(2) If the correspondence of atoms in g_i is not unique, we have to scan all permutations of individual subgroups of g_i. This added complexity is captured by functions σ' and χ.

The rest of the proof is straightforward. □

6. Algorithm

This section gives a summary and a pseudo-code of the algorithm for pairing atoms in protein structures.

The steps that need to be performed in order to find the best pairing are these:

(1) Find the best grouping g_A and g_B of atoms of structures A and B.
(2) Match *top-level* and *bottom-level* groups of g_A and g_B.
(3) Return the paring that yields the lowest value of RMSD.

6.1. Pseudo-code. Pseudo-codes 6.1, 6.2, 6.3, and 6.4 give a more detailed description of the algorithms presented in this chapter. We allow ourselves a certain level of abstraction, for example with function unpair that implicitly produces two structures from the pairing of atoms (because function $\mathrm{superimpose}$ was defined for a pair of structures).

7. Running Time and Space Complexity

The running time of the algorithm is $O(\mathrm{size}(g)T_n(\mathrm{superimpose}))$ where g is the grouping used and n is the number of atoms that are being superimposed and $\mathrm{size}(g)$ is defined in theorem 6.1. As we saw in the previous section, the time complexity of $\mathrm{superimpose}$ is $T_n(\mathrm{superimpose}) = O(n)$, therefore, the final complexity is $O(\mathrm{size}(g)n)$.

The groupings can be computed in $O(n \log(n))$ time by ordering the atoms in both structures (which takes $O(n \log(n))$ time) and then grouping the atoms together in several (but a fixed number) of linear passes.

The space complexity of the algorithm is $O(n)$, where n is the number of atoms.

Algorithm 6.1 Function best-fit(A, B).

Input: Structures A and B with their coordinates with geometric center as origin.
Output: Structure A superimposed onto structure B

$(A', B') \leftarrow \text{grouping}(A, B)$
$(R, \_) = \text{search}(A', B', \emptyset, 1)$
$A'' \leftarrow \text{inverseGrouping}_{B'}(R(A'))$
return A''

Algorithm 6.2 Function $\text{grouping}(A, B)$.

Input: Structures A and B.
Output: Two level grouping of atoms in both structures.

$A' = \text{groupTopLevel}(\text{groupBottomLevel}(A, \lambda x.(\tau_A(x), \rho_A x), \lambda x.\beta_A(x)), \lambda x.\rho_A(x))$
$B' = \text{groupTopLevel}(\text{groupBottomLevel}(B, \lambda x.(\tau_B(x), \rho_B x), \lambda x.\beta_B(x)), \lambda x.\rho_B(x))$
if $A' \equiv B'$ **then**
    return (A', B')
end if
$A' = \text{groupTopLevel}(\text{groupBottomLevel}(A, \lambda x.(\tau_A(x), \rho_A x), \lambda x.\beta_A(x)), \lambda x.x))$
$B' = \text{groupTopLevel}(\text{groupBottomLevel}(B, \lambda x.(\tau_B(x), \rho_B x), \lambda x.\beta_B(x)), \lambda x.x))$
if $A' \equiv B'$ **then**
    return (A', B')
end if
$A' = \text{groupTopLevel}(\text{groupBottomLevel}(A, \lambda x.\beta_A(x), \lambda x.\beta_A(x)), \lambda x.x))$
$B' = \text{groupTopLevel}(\text{groupBottomLevel}(B, \lambda x.\beta_A(x), \lambda x.\beta_B(x)), \lambda x.x))$
if $A' \equiv B'$ **then**
    return (A', B')
end if
Error: no pairing found.

Algorithm 6.3 Function $\text{search}(A, B, p, \text{groupIndex})$.

Input: Groupings of structures A and B, the "current" pairing p, and the index groupIndex of the group we are in.
Output: A 3x3 rotation matrix representing the transformation of structure A and the value of RMSD.

if $\text{groupIndex} \leq |A|$ **then**
    $B' \leftarrow B_{\text{groupIndex}}$
    $\text{rmsd} \leftarrow \infty$
    for each $A' \in \text{permutations}(A_{\text{groupIndex}})$ **do**
        $S_A \leftarrow \emptyset$
        for $i = 1$ **to** $|A'|$ **do**
            $S_A \leftarrow S_A \cup \text{matchSubgroup}(A'_i, B'_i)$
        end for
        $p' \leftarrow p \cup sp$
        $(R', \text{rmsd}') \leftarrow \text{search}(A, B, p', \text{groupIndex} + 1)$
        if $\text{rmsd}' < \text{rmsd}$ **then**
            $\text{rmsd} \leftarrow \text{rmsd}'$
            $R \leftarrow R'$
        end if
    end for
    return (R, rmsd)
else
    return $\text{superimpose}(\text{unpair}(p))$
end if

Algorithm 6.4 Function matchSubgroup(S_A, S_B).

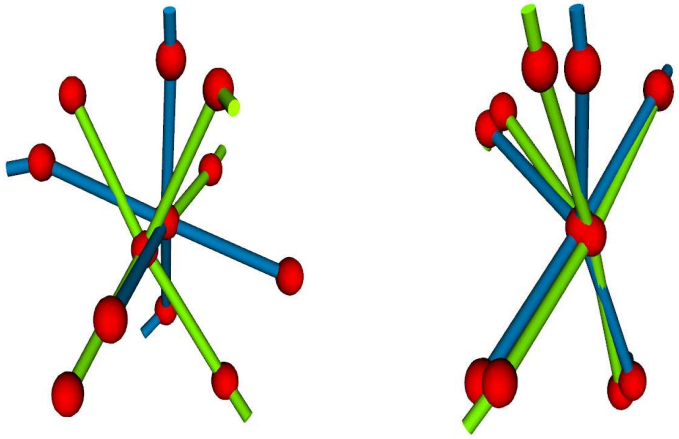
Input: Groups of atoms S_A and S_B.
Output: Pairing of atoms in S_A and S_B

```
pairing ← ∅
rmsd ← ∞
for i = 1 to |S_A| do
    if |S_A| = 1 then
        pairing ← pairing ∪ {((S_A)_i)_1, ((S_B)_i)_2}
    else
        for each A ∈ permutations((S_A)_i) do
            A' ← A
            B' ← (S_B)_i
            (_, rmsd') ← superimpose(A', B')
            if rmsd' < rmsd then
                rmsd ← rmsd'
                p ← p', where p' is the "identity" pairing of A' and B'
            end if
        end for
    end if
    pairing ← pairing ∪ p
end for
return pairing
```

8. Special Cases

For some binding sites, such as CaO_5, the *residue name* and *residue* groupings (assuming they are possible) do not always provide the best possible result. This is because of the wild nature of these binding sites – usually, each oxygen bound to the central calcium atom is on a residue with different name (the atom comes from different amino acid).

We can overcome this problem by forcing *element symbol* grouping. The result is illustrated by figures 6(a) and 6(b).



(a) Residue name. (b) Element symbol.

FIGURE 6.6. CaO_5 binding site superimposed using different groupings.

To identify this problem we have two options:

- Identify the problem visually.
- When superimposing many structures with different types of possible groupings at once and considering the "quality of alignment" as discussed in the next chapter (section 3), this problem can be identified when a negative error occurs.

9. Other Types of Groupings

There are several possible ways to extend the grouping scheme presented in this chapter.

A two level grouping is used in this thesis. However, it is possible to use entire hierarchy of groupings. For example we might add another level based on the quaternary structure of a protein. This would be particularly useful when dealing with very large structures such as whole proteins – i.e. when fitting two large proteins using the metals in their binding sites. A typical protein complex might contain eight *Ca* binding sites, two per each of four sub-complexes. Using the naive approach, we would end up with $8! = 40320$ paring search space size. However, using the information about quaternary structure we might reduce the search space size to $4!2! = 48$.

Furthermore, we are basing the grouping types entirely on the structural information of proteins provided a priori, omitting the geometry of the structure. It might be possible to derive a much more intricate grouping scheme based also on the geometry and atom bonds of the molecule. As a consequence, the algorithms provided in this thesis would work effectively for arbitrary 3D structures, not just proteins. This would have a variety of applications, for example in drug design.

Finally, when pairing atoms, we always assume that an atom with element symbol X has to be paired with an atom also with element symbol X. Nevertheless, in some cases, it might be desirable to for example pair oxygen and sulphur atoms. This can be easily achieved by considering an arbitrary equivalence relation on element symbols, at the cost of increased search space.

CHAPTER 7

Superimposing Multiple Structures

In this chapter we introduce an algorithm for superimposing multiple structures. The goal is to minimize the value of RMSD as defined for multiple structures in definition 4.4:

$$\mathrm{RMSD}(S) = \sqrt{\binom{m}{2}^{-1} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{RMSD}(S_i, S_j)^2},$$

where $m = |S|$ is the number of structures.

Ideally, we would like the solution to be as close to *pair-wise RMSD* as possible. This is captured by the following definition.

Definition 7.1. *We define function* $\mathrm{RMSD}_{\mathrm{opt}} : 2^{\mathcal{S}} \to \mathbb{R}$ *for a set of* $m = |S|$ *structures as*

$$\mathrm{RMSD}_{\mathrm{opt}}(S) = \sqrt{\binom{m}{2}^{-1} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{RMSD}_{\text{best-fit}}(S_i, S_j)^2}.$$

Unfortunately, as we will see below, achieving this value is, in general, impossible. Nevertheless, the value of $\mathrm{RMSD}_{\mathrm{opt}}$ will serve us when estimating how good a given superimposition is.

What we are looking for is a family T of transformations T_i with T_1 being the identity transformation (one of the structures needs to be "fixed") that minimizes the expression

$$T = \arg\min_{\{\phi_i\}} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{RMSD}(\phi_i(S_i), \phi_j(S_j))^2. \tag{7.1}$$

As in chapter 5, T_i can be written as $T_i(x) = R_i(x) + t_i$, the rotation and translation component. We know the optimal translation is zero if the structures are given in the coordinates with the geometrical center as origin (see the first section of chapter 5). Therefore, our task simplifies into finding a family R of rotations R_i (with R_1 being identity) for which

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{RMSD}(R_i(S_i), R_j(S_j))^2 \tag{7.2}$$

is minimized.

The algorithm presented here is in essence identical to that published in **[16]**, that, unlike this thesis, considers weighted RMSD. The algorithm was independently discovered by the author as well. However, there are two major differences from the algorithm presented in **[16]**. The first is that we consider the relation of the obtained result to the value $\mathrm{RMSD}_{\mathrm{opt}}$. The second is that pairing of atoms in not known a priori (which is assumed by the algorithm in **[16]**).

The concept of average structure (defined in 4.10) plays the central role in the algorithm. Moreover, the average structure serves as a *model* for the family of structures we seek to superimpose (in our case, usually a binding site of a protein).

1. Non-existence of the Optimal Solution

As mentioned above, the optimal solution does not, in general, exist. The following example illustrates why.

Example 7.1. *In this example a superimposition of three structures (figure 7.1) will be shown and various aspects of the problem will be demonstrated.*

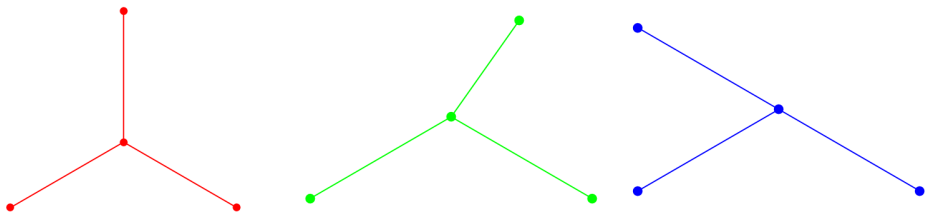


FIGURE 7.1. Three simple structures.

Figure 7.2 shows the first two structures superimposed after the application of pairwise superimpose algorithm.

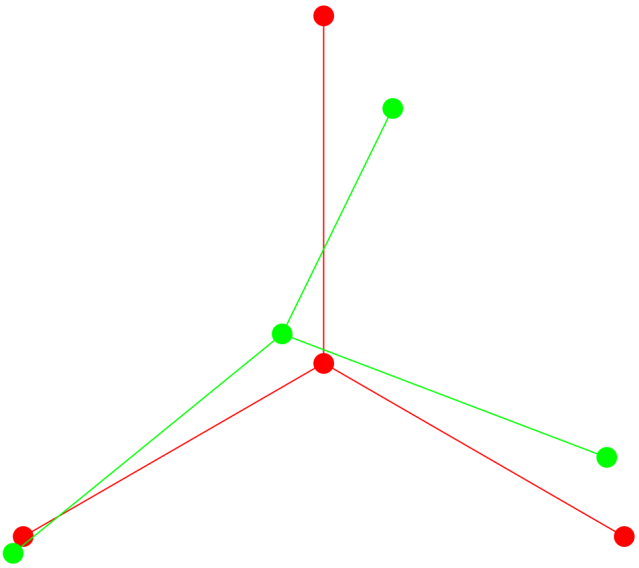


FIGURE 7.2. Superimposition of first two structures.

As we seen in chapter 5, the pairwise superimposition yields a unique result (up to the transformation of the whole system and some degenerate cases). Let us assume that the multiple superimposition can achieve the value $\mathrm{RMSD}_{\mathrm{opt}}$*. Then, obviously, all pairs of structures must be in the "optimal" mutual position. Therefore, aligning the third (blue) structure to the aligned versions of either the red or green structure should produce the same rotation of the original structure. Nevertheless, as we can see on figure 7.3, this is not the case. This means that, in general, the value* $\mathrm{RMSD}_{\mathrm{opt}}$ *cannot be achieved.*

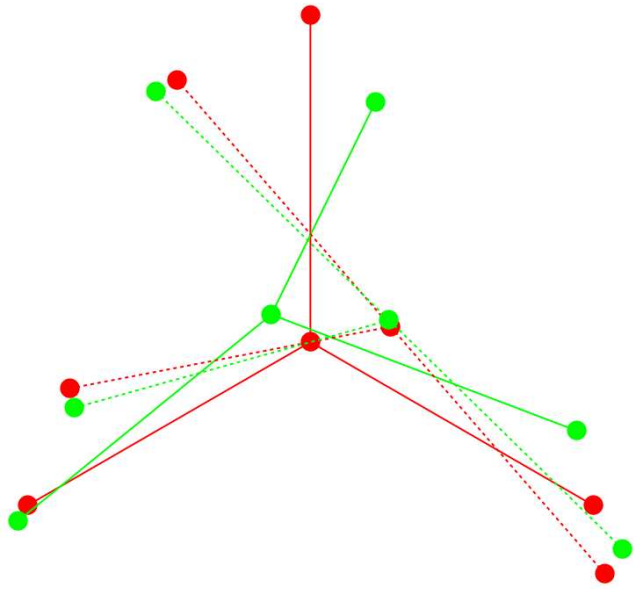


FIGURE 7.3. The third structure (dotted) superimposed to the first and then to the second one.

This is very unfortunate. The question remains, how should we align the third structure in order to achieve the best result. Figure 7.4 shows what happens if we superimpose the blue structure and the average structure of the first two (red and green) aligned structures (dotted red and green structures are the blue structure superimposed to red and then green as in figure 7.3). Aligning to the red structure yields an overall RMSD of 0.359 Å*, to the green one* 0.360 Å *and finally aligning to the average of the red and green (depicted by dotted brown) yields* 0.350 Å*. As we can see, the "average" case is the best one.*

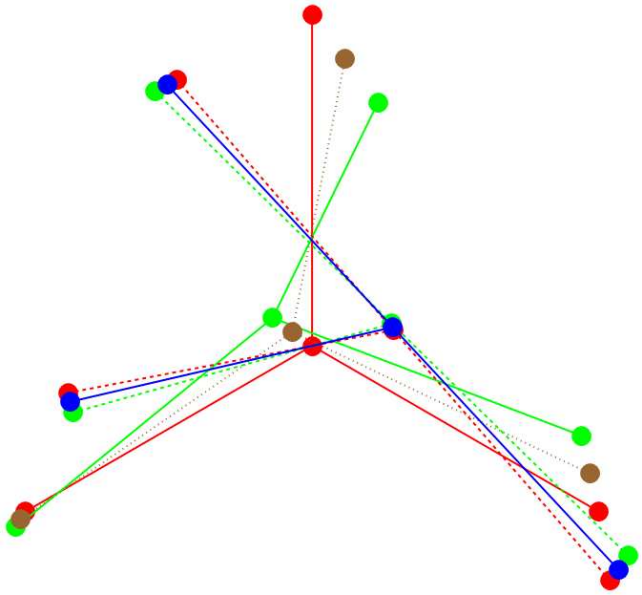


FIGURE 7.4. The third structure (blue) superimposed to the average of the other structures.

Nevertheless, even after aligning the blue structure to the average of the first two ones, we can achieve a better result. Superimposing all three structures to their average (i.e. the average structure of the aligned versions of red, green and blue structures) provides a better result: 0.345 Å*. Superimposing structures to their average is the core idea behind the algorithm presented in this chapter.*

2. Towards the Algorithm

The basic idea behind the algorithm is: Iteratively superimpose all structures to their (current) average structure until we get an *acceptable* result.

The following text provides the reader with lemmas and theorems supporting the above idea.

First, we show that $\mathrm{RMSD}_{\mathrm{opt}}$ is the best result we can hope to achieve.

Theorem 7.1. *Let $S \in 2^{\mathcal{S}}$. Then $\mathrm{RMSD}_{\mathrm{opt}}(S) \leq \mathrm{RMSD}_{\text{best-fit}}(S)$, where $\mathrm{RMSD}_{\text{best-fit}}$ is the result provided by any algorithm superimposing structures S.*

PROOF. This follows directly from the fact, that a pair of structures $A, B \in \mathcal{S}$ cannot be "closer" than $\mathrm{RMSD}(\text{best-fit}(A, B), B)$. □

The following lemma and theorem have been taken (and modified) from **[16]**.

Lemma 7.1. *For any set of points $\{p_i\}$ of size n, the sum of squared distance from $p_1, p_2, ..., p_n$ to any point q equals the sum of distances to the average point $\overline{p} = \frac{1}{n}\sum_{i=1}^{n} p_i$ plus the sum from $\overline{p}$ to q:*

$$\sum_{i=1}^{n} \|p_i - q\|^2 = \sum_{i=1}^{n} \|p_i - \overline{p}\|^2 + \sum_{i=1}^{n} \|q - \overline{p}\|^2.$$

PROOF. We subtract the second term from both sides, expand the difference of squares, and then apply the definition of $\overline{p}$ in the penultimate step:

$$\begin{aligned}
&\sum_{i=1}^{n} \left(\|p_i - q\|^2 - \|p_i - \overline{p}\|^2\right) \\
&\quad = \sum_{i=1}^{n} (p_i - q + p_i - \overline{p}) \cdot (p_i - q - p_i + \overline{p}) \\
&\quad = (\overline{p} - q) \cdot \sum_{i=1}^{n} (2p_i - \overline{p} - q) \\
&\quad = (\overline{p} - q) \cdot \sum_{i=1}^{n} (\overline{p} - q) = \sum_{i=1}^{n} \|q - \overline{p}\|^2.
\end{aligned}$$

□

The next theorem says that if the RMSD is used to compare multiple structures, we can establish the value of the RMSD by comparing the structures to their average structure. By comparing to the average structure, we reduce the number of pairs of structures that must be compared from $m(m-1)/2$ to m (where m is the number of structures).

Theorem 7.2. *Let $S = \{S_i\}$ be the set of m structures with n points. Then the sum of squared distances for all pairs (RMSD) equals the sum of squared distances to the average structure $\overline{S}$:*

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{RMSD}(S_i, S_j)^2 = m \sum_{i=1}^{m} \mathrm{RMSD}(S_i, \overline{S})^2$$

PROOF. First, we have

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{RMSD}(S_i, S_j)^2 = \frac{1}{n} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{k=1}^{n} \|S_{ik} - S_{jk}\|^2$$

and

$$\sum_{i=1}^{m} \mathrm{RMSD}(S_i, \overline{S})^2 = \frac{1}{n} \sum_{i=1}^{m} \sum_{k=1}^{n} \left\|S_{ik} - \overline{S}_k\right\|^2.$$

In lemma 7.1, replace q by S_{jk}, p_i by S_{ik}, and $\overline{p}$ by $\overline{S}_k$. Then multiply by m, and sum over all j and k to obtain

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\|S_{ik}-S_{jk}\|^2 = 2\sum_{k=1}^{n}\sum_{i=1}^{m}\sum_{j=1}^{m}\|S_{ik}-\overline{S}_k\|^2.$$

We can rearrange the order of summation on the left, noticing that terms with $i \neq j$ cancel and every other term appears twice:

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\|S_{ik}-S_{jk}\|^2 = 2\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{k=1}^{n}\|S_{ik}-S_{jk}\|^2.$$

The resulting equation gives the desired result after dividing out the extra factor of two:

$$\begin{aligned}
&2\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{k=1}^{n}\|S_{ik}-S_{jk}\|^2 \\
&= 2\sum_{i=1}^{m}\sum_{k=1}^{n}\left(m\|S_{ik}-\overline{S}_k\|^2\right) \\
&= 2m\sum_{i=1}^{m}\sum_{k=1}^{n}\|S_{ik}-\overline{S}_k\|^2.
\end{aligned}$$

□

Finally, we show that by superimposing structures to their average the value of RMSD is improved. This ensures that the algorithm converges, most likely towards the global minimum.

Theorem 7.3. *Let S be the set of m structures with n points. Then superimposing each of the structures to their average structure $\overline{S}$ improves the value of RMSD:*

$$\mathrm{RMSD}(S') \leq \mathrm{RMSD}(S),$$

where $S'_i = \mathrm{superimpose}(S_i, \overline{S})$.

PROOF. This is a direct consequence of theorem 7.2.
More specifically, in the sum

$$\sum_{i=1}^{m}\mathrm{RMSD}(S_i,\overline{S})^2$$

we have

$$\mathrm{RMSD}(\mathrm{superimpose}(S_i,\overline{S}),\overline{S}) \leq \mathrm{RMSD}(S_i,\overline{S})$$

for each i. □

A careful reader might have noticed that in the previous theorem we have used the function superimpose instead of best-fit. The reason for this is that by repeatedly superimposing structures to their average, the order of atoms in individual structures can change. This results in better RMSD of the affected structure to the average one, nevertheless, might result in a worse RMSD value for the whole ensemble of structures.

Fortunately, as testing shows this "change of ordering" only happens when the algorithm converges close to the "final" solution, therefore the RMSD value for the whole ensemble does not get affected much. Therefore, the iteration of the algorithm can be terminated once a *worse* solution has been produced than in the previous step.

3. Quality of the Alignment

In order to establish the quality of the alignment, absolute and relative *errors* are defined.

Definition 7.2. *We define the absolute error of alignment of structures $S \in 2^{\mathcal{S}}$ as the difference*

$$\mathrm{ERR}_{\mathrm{abs}} = \mathrm{RMSD}_{\text{best-fit}}(S) - \mathrm{RMSD}_{\mathrm{opt}}(S).$$

Definition 7.3. *We define the relative error of alignment of structures $S \in 2^{\mathcal{S}}$ as the difference*

$$\mathrm{ERR}_{\mathrm{rel}} = \frac{\mathrm{ERR}_{\mathrm{abs}}}{\mathrm{RMSD}_{\mathrm{opt}}(S)} = \frac{\mathrm{RMSD}_{\text{best-fit}}(S) - \mathrm{RMSD}_{\mathrm{opt}}}{\mathrm{RMSD}_{\mathrm{opt}}(S)}.$$

Note that these *errors* are not errors in the traditional sense of the word, because as we have seen in the example of this chapter, the optimal alignment does not always exist.

When modeling a binding site, values of $\mathrm{ERR}_{\mathrm{abs}}$ and $\mathrm{ERR}_{\mathrm{rel}}$ help to establish two important things:

- Indicates the quality of the model of the given site.
- As mentioned in the previous chapter (in section 8), the error can actually take on negative values. This is of course in contradiction with theorem 7.1.

  The problem is in the "grouping" part of the algorithm for pairwise superimposition. When superimposing to the average structure, different grouping, and thus pairing, can be established than when the structures are compared directly. This indicates that the grouping type must be manually overridden in order to provide a better result. Unfortunately, this override usually comes at the cost of computation time, therefore, often the "default" results are good enough for any practical application and the negative error can be ignored.

4. Algorithm

The algorithm works by iteratively superimposing structures to their average structure. Also, one extra iteration superimposing structures to the first of them is also needed to establish an initial pairing of the atoms. This step is present because we need to find an approximation of pairing of atoms for all structures in order to compute a "meaningful" initial average structure.

The outline of the algorithm is:

(1) Superimpose all structures to the first one to ensure good approximation of average structure.
(2) Compute the value q as the sum of RMSD of each structure to the average structure $\overline{S}$.
(3) Superimpose all structures to $\overline{S}$.
(4) Compute the value q' as the sum of RMSD of each structure to the average structure $\overline{S}$.
(5) If $\delta = (q - q')/q > \epsilon$, then go to step 2, otherwise stop and return the current alignment and the average structure.

In step 5, the value of $\delta = (q - q')/q$ is the percentage of the change of RMSD between two consecutive steps and $\epsilon > 0$ is a fixed parameter determining when the alignment is considered good enough.

By an iteration of the algorithm we understand performing steps 2 to 5. Step 1 is also considered as a separate iteration.

4.1. Establishing the Quality of the Alignment. The basic algorithm can also be extended by computing the values of $\mathrm{RMSD}_{\mathrm{opt}}$, $\mathrm{ERR}_{\mathrm{abs}}$, and $\mathrm{ERR}_{\mathrm{rel}}$:

(1) Execute the "basic" version of the algorithm.
(2) Compute the values of $\mathrm{RMSD}_{\mathrm{opt}}$, $\mathrm{ERR}_{\mathrm{abs}}$, and $\mathrm{ERR}_{\mathrm{rel}}$.
(3) Return the alignment, average structure, and values computed in the previous step.

4.2. Pseudo-code. The following pseudo-code provides the reader with a more detailed description of the algorithm:

Algorithm 7.1 Function $\mathrm{superimpose}(S)$.

Input: A set of m structures with their coordinates with the geometric center as origin.
Output: Mutually superimposed structures and their average structure.

for $i = 2$ **to** m **do**
  $S_i \leftarrow \text{best-fit}(S_i, S_1)$
end for
$\delta \leftarrow \infty$
while $\delta \geq \epsilon$ **do**
  $q \leftarrow \mathrm{rmsdToAverage}(S)$
  $\overline{S} \leftarrow \mathrm{average}(S)$
  for $i = 1$ **to** m **do**
    $S_i \leftarrow \text{best-fit}(S_i, \overline{S})$
  end for
  $q' \leftarrow \mathrm{rmsdToAverage}(S)$
  $\delta \leftarrow (q - q')/q$
end while
return $(S, \mathrm{averageStructure}(S))$

Algorithm 7.2 Function $\mathrm{superimposeWithErrorEstimation}(S)$.

Input: A set of m structures and a value determining the quality of the result we want to achieve.
Output: Mutually superimposed structures, average structure, and information about the quality of the alignment.

$(S', \overline{S}) \leftarrow \mathrm{superimpose}(S)$
$\mathrm{rmsd}_{\mathrm{opt}} \leftarrow \sqrt{\frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{RMSD}(\text{best-fit}(S_i, S_j), S_j)^2}$
$\mathrm{ERR}_{\mathrm{abs}} \leftarrow \mathrm{RMSD}(S') - \mathrm{rmsd}_{\mathrm{opt}}$
$\mathrm{ERR}_{\mathrm{rel}} \leftarrow \mathrm{ERR}_{\mathrm{abs}} / \mathrm{rmsd}_{\mathrm{opt}}$
return $(S', \overline{S}, \mathrm{ERR}_{\mathrm{abs}}, \mathrm{ERR}_{\mathrm{rel}})$

Algorithm 7.3 Function $\mathrm{rmsdToAverage}(S)$.

Input: A set of m structures with n atoms.
Output: A sum of RMSD to the average structure

$\overline{S} \leftarrow \mathrm{averageStructure}(S)$
$\mathrm{rmsd} \leftarrow \sum_{i=1}^{m} \mathrm{RMSD}(S_i, \overline{S})$
return rmsd

Algorithm 7.4 Function $\mathrm{averageStructure}(S)$.

Input: A set of m structures with n atoms.
Output: Average structures.

for $i = 1$ **to** m **do**
  $\overline{S}_i \leftarrow \frac{1}{n} \sum_{j=1}^{m} (S_i)_j$
end for
return $\overline{S}$

5. Running Time and Space Complexity

The basic version of the algorithm runs in time $\Omega(mT(n))$ where m is the number of structures being aligned and $T(n)$ is the time required to superimpose (and find the best pairing of atoms) of a pair of structures (see section 7 of the previous chapter). The lower estimate is used because it is difficult to upper-bound the number of iterations needed before the algorithm converges. Nevertheless, it seems that the actual running time is very close to $\Theta(mT(n))$ (this claim is supported by results from chapter 14 and the proof is left as a future work). Finally, the space complexity is $O(mn)$.

The quality establishing version of the algorithm runs in time $\Omega(m^2T(n))$, because each pair of structures has to be superimposed independently in order to establish the value of $\mathrm{RMSD}_{\mathrm{opt}}$. The space complexity is again $O(mn)$.

6. Modeling Binding Sites

The average structure might serve as a model for the particular binding site. Such a model can be used for example to predict binding sites in proteins or to derive interesting properties of structures, i.e. the properties of ZnN_2S_2 binding site shown in section 14.2.

Part 3

Methods

CHAPTER 8

PDB File Format

The Protein Data Bank (PDB) file format **[19]** is a textual file format describing the three dimensional structures of molecules held in the Protein Data Bank **[20]**. Most of the information in that database pertains to proteins, and the PDB format accordingly provides for a rich description and annotation of protein properties. However, proteins are often crystallized in association with other molecules such as water, ions, nucleic acids, drug molecules and so on, which therefore can be described in the PDB format as well.

Every PDB file is presented in a number of lines and each line in the PDB entry file consists of 80 columns. The first six columns of every line contain a record name that is left-justified and separated by a blank. Record names are predefined such as `HEADER`, `AUTHOR`, `MODEL`, `ATOM`, and `HETATM`.

It is the last two record types we are interested in within this thesis the most as they provide information about atom positions, residues, etc. `ATOM` entries describe the coordinates of the atoms that are part of the protein, whereas as `HETATM` entries describe coordinates of hetero-atoms – atoms that are not part of the protein molecule.

In more detail, `ATOM` and `HETATM` entries contain the following information about atoms:

Atom serial number: An integer.
Atom name: Four characters.
Element symbol: Two characters.
Residue name: A string.
Residue sequence number: An integer.
Chain identifier: A character.
Atom position: Three real numbers.
Alternate location indicator: A character.
Other information: Code for insertion of residues, occupancy, temperature factor, segment identifier, and charge on the atom.

Examples of *ATOM* entries are:

```
ATOM   1276  N   CYS B 172      18.801  29.113  30.970  1.00 20.52           N
ATOM   1277  CA  CYS B 172      19.724  28.014  30.655  1.00 22.50           C
ATOM   1278  C   CYS B 172      20.153  28.103  29.195  1.00 22.05           C
ATOM   1279  O   CYS B 172      19.722  29.011  28.487  1.00 21.88           O
ATOM   1280  CB  CYS B 172      19.049  26.681  30.926  1.00 22.80           C
```

And an examples of *HETATM* entries are:

```
HETATM 2410 ZN    ZN B 250      15.939  27.351  29.688  1.00 28.93          ZN
HETATM 3621  C1  GYP A 238      33.024 128.964 117.888  1.00 23.39           C
HETATM 3628  O1  GYP A 238      33.900 129.324 118.989  1.00 25.26           O
HETATM 3649 MN    MN B 239      14.515  87.283  71.760  1.00 14.25          MN
HETATM 3650 CA    CA B 240      13.201  83.977  73.969  1.00 13.36          CA
```

Usually, the naming convention for PDB files is *xxxx.pdb*, where *xxxx* is a four letter identifier of a particular protein, for example 1a5t. In this thesis we are not interested in the whole proteins, but rather only small subparts of them (such as binding sites). Therefore, an extended convention matching the pattern *xxxx_yyyy.pdb* is used. Again, *xxxx* is the identifier of the protein, whereas *yyyy* corresponds to the *atom serial number* of the first atom in the motif.

CHAPTER 9

Microsoft .NET Framework and DirectX

1. .NET Framework

The Microsoft .NET Framework [**21**] is used as the runtime library for SiteBinder, the software implementing the algorithms presented in this thesis.

The Microsoft .NET Framework is a software framework that can be installed on computers running Microsoft Windows operating systems. It includes a large library of coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework is a Microsoft offering and is intended to be used by most new applications created for the Windows platform.

Programs written for the .NET Framework execute in a software environment that manages the program's runtime requirements. Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security, memory management, and exception handling. The class library and the CLR together constitute the .NET Framework.

Version 3.0 of the .NET Framework is included with Windows Server 2008 and Windows Vista. The current stable version of the framework, which is 3.5, can also be installed on Windows XP and the Windows Server 2003 family of operating systems.

1.1. Mono. One drawback of Microsoft .NET Framework is that it is for Windows operating systems only. Fortunately, there is an alternative called Mono.

Mono [**22**] is a free and open source project led by Novell (formerly by Ximian) to create an Ecma standard compliant, .NET-compatible set of tools, including among others a C# compiler and a Common Language Runtime. Mono can be run on Linux, BSD, UNIX, Mac OS X, Solaris and Windows operating systems.

Porting a native C# code .NET application to Mono is usually a straightforward task. Nevertheless, the graphical user interface is another issue and often needs to be redone.

2. DirectX

For the rendering of molecular structures in SiteBinder, Microsoft Direct3D (which is a component of DirectX) is used.

Microsoft DirectX [**23**] is a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms. Originally, the names of these APIs all began with Direct, such as Direct3D, DirectDraw, DirectMusic, DirectPlay, DirectSound, and so forth. The name DirectX was coined as shorthand term for all of these APIs (the X standing in for the particular API names) and soon became the name of the collection.

Direct3D (the 3D graphics API within DirectX) is widely used in the development of video games for Microsoft Windows, Microsoft Xbox, and Microsoft Xbox 360. Direct3D is also used by other software applications for visualization and graphics tasks such as CAD/CAM engineering.

2.1. OpenGL. Similarly to Microsoft .NET, DirectX is also available only for Windows operating systems. Fortunately, should the need arise, OpenGL can be used in its place.

OpenGL (Open Graphics Library) [**24**] is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The

interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms.

CHAPTER 10

Eigenvalues and Eigenvectors Computation

Computing eigenvalues and eigenvectors is a crucial part of the algorithm for superimposing two three-dimensional structures described in chapter 5. Because of this reason, an already existing and tested implementation was used. It is called ALGLIB.NET – a multilingual collection of algorithms designed to solve problems in the field of numeric analysis and data processing [**25**].

The following text taken from [**26**] gives an overview of existing algorithms that solve the eigenvalue and eigenvector problem.

1. Overview of the Existing Algorithms

The first algorithm solving the eigenvalue problem for a symmetric NxN matrix was the Jacobi algorithm which had reduced a matrix to diagonal form by using an orthogonal transformation. During the transformations, the diagonal elements were increased, and the off-diagonal elements were decreased. The result of this process is a matrix whose off-diagonal elements were equal to 0, and whose diagonal elements were equal to the eigenvalues.

The Jacobi algorithm is simple but ineffective: it performs operations upon a full matrix A even when most of the elements have already been converged to 0. Almost all later algorithms for solving the symmetric eigenvalue problem preliminary reduce the matrix to tridiagonal form (this operation is performed by non-iterative algorithm in a finite number of steps) and then work with a tridiagonal matrix.

The most widespread algorithms family is a algorithms based on QL/QR iteration applied to a tridiagonal matrix. We can mention the algorithm from the LINPACK [**27**] library which implements the simplest QL algorithm (the subroutines which are related to this algorithm could be found in many sources) and a more up-to-date variant from the LAPACK [**28**] library (the xSTEQR subroutine) which uses implicit shifts and can switch between QL and QR iterations depending on their performance for the given matrix. The algorithm from the LAPACK library is bigger but more reliable and accurate, so it is this algorithm that is used as the basis of a source code available on this page.

There are some other algorithms for finding the eigen pairs in the LAPACK library. We can point to a divide-and-conquer algorithm and an RRR algorithm. They can significantly speed up the finding of eigen pairs for the big symmetric tridiagonal matrix. Speeding-up can reach several dozen times for a tridiagonal matrix, for a symmetric matrix (taking into account the time required to reduce the matrix to tridiagonal form) it can reach 2-4 times. These algorithms are rather complex, therefore they have not been included in the ALGLIB library yet.

CHAPTER 11

Software for Superimposing and Visualizing Molecules

This chapter lists several software packages capable of visualizing and superimposing molecular structures. Summary of the software is shown in table 11.1.

VMD: VMD **[29]** is a molecular graphics program designed for the interactive visualization and analysis of biopolymers such as proteins, nucleic acids, lipids, and membranes. VMD runs on all major Unix workstations, AppleMacOS X, and Microsoft Windows. At its heart, VMD is a general application for displaying molecules containing any number of atoms. Beside this VMD can be used for interactive molecular dynamics simulations on a remote supercomputer or high-performance workstation. It provides many commands for molecular analysis and in addition, Python and Tel scripting languages are embedded for processing text commands. VMD is capable of superimposing pairs of molecules.

PyMOL: PyMOL **[30]** is an open-source, user-sponsored, molecular visualization system created by Warren Lyford DeLano and commercialized by DeLano Scientific LLC, which is a private software company dedicated to creating useful tools that become universally accessible to scientific and educational communities. It is well suited to producing high quality 3D images of small molecules and biological macromolecules such as proteins. According to the author, almost a quarter of all published images of 3D protein structures in the scientific literature were made using PyMOL.

PyMOL can also be used for superimposing multiple molecular structures. Before superimposing molecules, PyMOL first automatically finds a subset of atoms the superimposition will be based on.

UCSF Chimera: UCSF Chimera **[31]** is a highly extensible program for interactive visualization and analysis of molecular structures and related data, including density maps, supramolecular assemblies, sequence alignments, docking results, trajectories, and conformational ensembles. High-quality images and animations can be generated. Chimera includes complete documentation and several tutorials, and can be downloaded free of charge for academic, government, non-profit, and personal use. Chimera is developed by the Resource for Biocomputing, Visualization, and Informatics and funded by the NIH National Center for Research Resources.

UCSF Chimera is capable of superimposing pairs of molecules.

Other software: The following software packages are capable of superimposing pairs or multiple molecules: Qmol **[32]**, Swiss-PdbViewer **[33]**, Gromacs **[34]**.

Name	Version	Pairwise	Pairing	Multiple	URL
SiteBinder	2.4.4	Yes	Yes	Yes	*ncbr.chemi.muni.cz/~dave/SiteBinder/*
VMD	1.8.7	Yes	No	No	*www.ks.uiuc.edu/Research/vmd/*
PyMOL	1.2r1	Yes	Limited	Yes	*pymol.sourceforge.net*
UCSF Chimera	1.4	Yes	No	No	*www.cgl.ucsf.edu/chimera/*
Qmol	4.02	Yes	No	No	*www.dnastar.com/qmol/*
Swiss-PdbViewer	4.0.1	Yes	No	No	*spdbv.vital-it.ch*
Gromacs	4.0.5	Yes	No	No	*www.gromacs.org*

TABLE 11.1. Software for superimposing molecules.

Part 4

Implementation

CHAPTER 12

Overview of SiteBinder

As a part of this thesis the presented algorithms were implemented in the software package called SiteBinder.

1. Features

Features of SiteBinder include:

Loading of PDB structures: Structures in PDB format can be loaded.
Superimposing: Superimposing of an arbitrary number of structures (limited by computer's memory and time constraints).
Calculation RMSD: RMSD of arbitrary number of molecules can be computed.
Modeling Binging Site: Models of binding sites can be computed.
Rendering: Structures are rendered using Microsoft Direct3D.
Highlighting: Various types of highlighting are supported to enable the user a more friendly manipulation with structures.
Exporting Results: Superimposed structures can be exported in PDB format. Also, structures can be rendered to an PNG image.

1.1. Versions. Two major versions of the software are available:

SiteBinder: An application with rich graphical user interface.
SiteBinder CMD: A simple command line interface to SiteBinder Core.

2. Programming Language

SiteBinder is implemented in C# [**35**]. The reasons for choosing this language are these:

LINQ: A query language build into C# that makes handling structures almost a trivial task. More on LINQ for example at [**36**].
Multi-platform: .NET Framework on Windows, Mono [**22**] on Windows/Linux/Mac/...
Microsoft Visual Studio: Rich features of Visual Studio 2008 (that supports C#) make the development of graphical user interface relatively easy task.

3. Running Environment

SiteBinder is a software package for Microsoft .NET Framework 3.5 and Microsoft DirectX (more specifically its 3D part called Direct3D) for displaying graphics. The current version works under all platforms supported by .NET Framework including Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008 and Windows 7.

4. Hardware Requirements

The hardware requirements are the same as of the operating system the software is running on. Furthermore, a graphic card that supports Direct3D is required (Direct3D support is fairly common in today's computers).

5. Input Specification

SiteBinder supports input from the PDB file format, however, some restrictions apply:

- There can be only one structure contained in one file.
- Alternate locations of atoms are not supported.

- If a file contains more chains, only one chain can be displayed and operated with at any given time.

In order for SiteBinder to work properly, these columns have to be filled:

Atom serial number: A unique integer identifier of the atom.
Atom name: String representing atom name.
Element symbol: Element symbol of the atom, for example S, *Ca*, etc. If *element symbol* is not filled, part of *atom name* record is used instead (the first two characters of the name).
Residue name: String representing name of the residue the atom is on, usually the name of an amino acid.
Residue sequence number: A unique integer identifier of the residue the atom is on.
Chain identifier: Usually represents the ternary structure of the protein.
Position: 3D coordinates of the atom.

Moreover, SiteBinder supports a special naming convention to make working with large a number of structures easier:

- If a given file has $m\_$ prefix, all atoms in the structure get automatically selected upon load.
- If there are files $m\_x.pdb$ and $ma\_x.pdb$ in the same directory, structure "x" will load from $ma\_x.pdb$ file and atoms from $m\_x.pdb$ will be selected.

6. Output Specification

By default the only output SiteBinder provides is a floating point number representing the resulting RMSD (or two numbers if we consider the error or multiple alignment).

Nevertheless, aligned structures can be exported to PDB files with these properties:

- $m\_x.pdb$ file is created for each structure and contains its selected atoms.
- $ma\_x.pdb$ file is created for each structure and contains all of its atoms.
- The order of atoms in all exported (and aligned) structures is the following:
  - First the selected atoms are stored.
  - 1st atom in structure "x" corresponds to the 1st atom in structure "y", etc.
  - Non-selected atoms are appended to the end of the file and sorted by their element symbol.
- Only `ATOM` and `HETATM` records are exported.
- Only the currently selected chain is exported.

7. User Interface

7.1. SiteBinder CMD. SiteBinder CMD has a very simple command line interface working as following:

```
sitebinder.cmd.exe s1.pdb s2.pdb ... si.pdb
```

All atoms in each structure are implicitly selected (the prefix conventions discussed above are ignored).

If the loaded structures can be superimposed, the output is a number representing RMSD, otherwise an error is reported.

7.2. SiteBinder Graphical User Interface. Figure 12.1 shows SiteBinder GUI with several structures loaded and selected. The structure of the main window is split into four basic parts:

Rendering Window: Shows the render of currently selected structures. Advanced features such as highlighting parts of a molecule are supported.
Structure List: Contains a list of structures and basic information about them, such as which residues they contain.
Atom Selection Tree: A tree-view allowing the user to select atoms on structures.
RMSD Panel: Provides value of RMSD of currently selected structures (assuming they were superimposed) and the error of the alignment (if available). Located under the Structure List and Atom Selection Tree

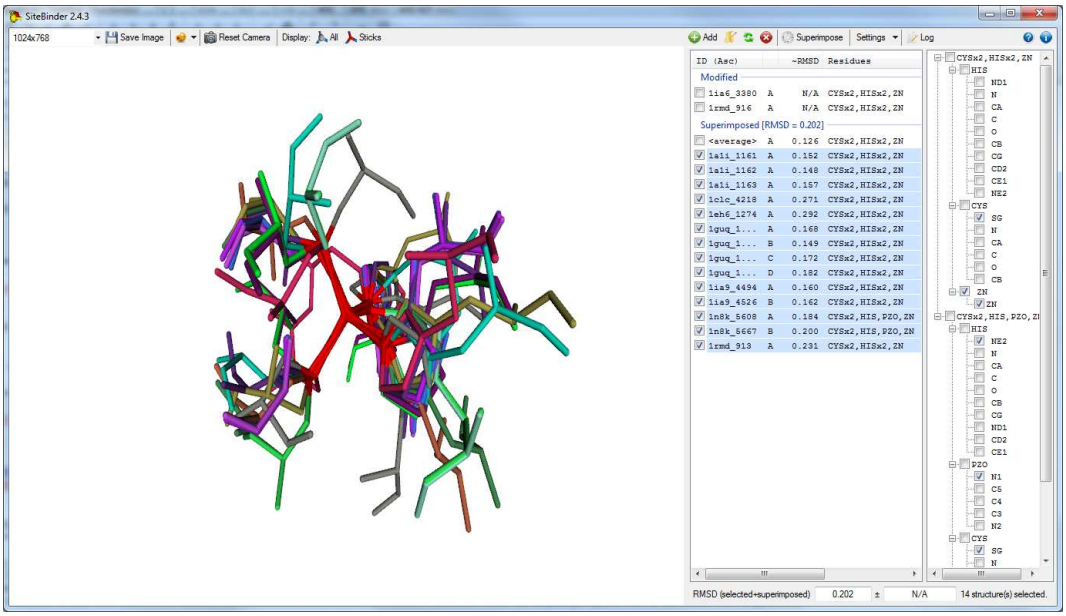


FIGURE 12.1. SiteBinder.

Furthermore, SiteBinder contains a *log window* allowing the user to see exceptions during the program runtime, and detailed information about each superimposition.

Each of the individual components is described in detail bellow.

7.2.1. *Structure List.* The structure list is shown on figure 12.2.

Information provided by the structure list:

Id: Identifier of the structure (extracted from the filename of the structure).

Chain Identifier: Currently displayed chain.

Average RMSD (~RMSD): Average RMSD to the other structures of the last superimpose operation.

Residues: Residue names of residues contained in the given structure.

Selected Atoms: Alphabetically ordered element symbols of selected atoms.

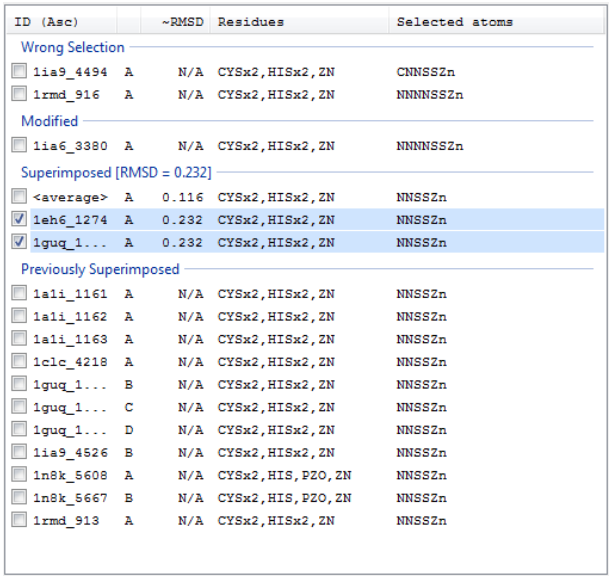


FIGURE 12.2. Structure list.

Structures are grouped into five categories:

Added: Structures that were added and no changes were made to them.

Modified: Structures that were changed (changed selection or chain).
Superimposed: Structures that are currently superimposed along with the current average structures.
Previously Superimposed: Structures that used to belong to *Superimposed* group.
Wrong Selection: Structures that could not be superimposed because their selections were not compatible.

If any modification is made to the structures in *Superimposed* group, the average structure is removed.

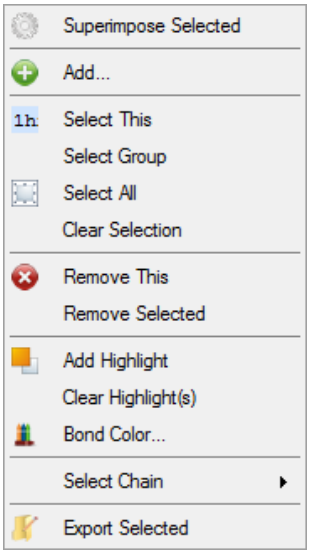


FIGURE 12.3. Structure list pop-up menu.

7.2.2. *Structure List Pop-up Menu.* The structure list pop-up menu (shown on figure 12.3) can be invoked by right clicking into a Structure List.

The following features are available:

Superimpose Selected: Superimposes the selected structures.
Add: Opens the add structure dialog.
Select This: Selects only the "clicked" structure and deselects all other structures.
Select Group: Selects all structures in the "clicked" structure's group. Average structure is excluded from this selection and has to be selected manually.
Select All: Selects all structures. Average structure is excluded from this selection and has to be selected manually.
Clear Selection: Deselects all structures.
Remove This: Removes the "clicked" structure.
Remove Selected: Removes all selected structures.
Highlight: Puts highlight on the "clicked" structure. This visually distinguishes the structure in the list.
Clear Highlight(s): Removes all active highlights.
Bond Color: Allows the user to change the color of the "clicked" structure.
Select Chain: Allows user to select a different chain on the "clicked" structure.
Export Selected: Exports all selected structures.

FIGURE 12.4. Structure manipulation toolbar.

7.2.3. *Structure Manipulation Toolbar.* The toolbar shown on figure 12.4 has the following features:

Add: Opens the add structure dialog.
Export: Exports all selected structures.
Reload: Reloads all structures.
Clear: Removes all structures.
Superimpose: Superimposes selected structures.
Settings: This feature is discussed in separate section bellow.
Log: Shows log.
Help: Shows help.
About: Shows about dialog.

7.2.4. *Settings.* The following settings can be adjusted:

Allow Reflection: Enables/disables the feature to look for improper rotations as well (see chapter 5 for more details). By default, this feature is disabled.

Method: Allows the user to select the method for the multiple alignment algorithm. One slower (running in quadratic time), proving information about quality of the alignment, and another faster (running in linear time), presenting only the value of RMSD (see chapter 7 for more details). By default, the basic version of the algorithm is selected.

Grouping Level: Allows the user to manually override the grouping level to sometimes achieve better results (see chapter 6). By default, *residue name* grouping is used.

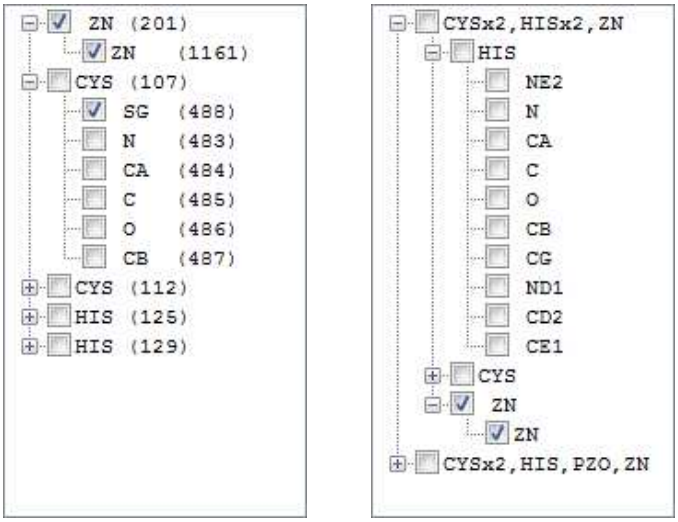


FIGURE 12.5. Atom selection for single (left) and multiple (right) structures.

7.2.5. *Atom Selection Tree.* SiteBinder supports a comfortable selection of atoms (shown on figure 12.5) for superimposing structures.

There are two modes of this function:

Single selection: A tree of atoms grouped by residue name is provided and individual selection of atoms is possible.

Multiple selection: An aggregate of all structures' atoms is computed based on the number of and types of residues they contains and atom names. This allows the selection of the same atoms over multiple structures. The structures are grouped by the types of residues they contain.

7.2.6. *Progress Dialog.* During the superimposition of structures, a progress dialog (shown on figure 12.6) is displayed. The dialog provides the following information:

- Number of current iteration and error of the last iteration.
- Progress of the current iteration.
- Progress of the current superimposition of pair of structures.
- Time elapsed since the operation has started.

Furthermore, the dialog gives the user the option to abort the process.

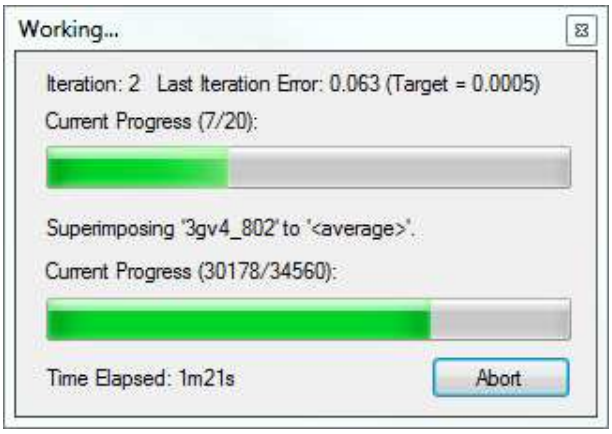


FIGURE 12.6. Progress dialog.

7.2.7. *Logs.* The logging dialog is used to display two types of information. One is error reports on exceptions during the program runtime with possible suggestions of solutions and the other is the report on alignment operations and their summary. The log is reset each time *Superimpose* is called.

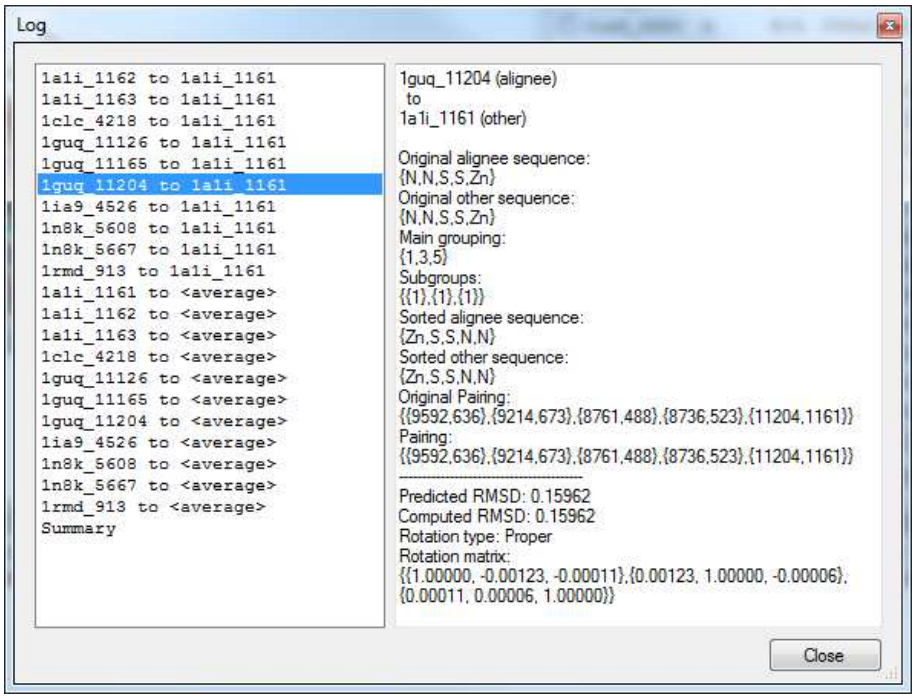


FIGURE 12.7. Log.

7.2.8. *Rendering Toolbar.* The rendering toolbar provides these features:

Resolution selection: allows the user to select the resolution of the image to be exported.
Save Image: Brings up the save image dialog.
Settings: Allows the user to select various colors and set up Direct3D.
Reset Camera: Resets the camera.
Display options: Allows the user to change the current display type as discussed above.

FIGURE 12.8. Rendering toolbar.

7.2.9. *Direct3D Settings.* Direct3D Settings dialog (shown on figure 12.9) allows the user to change the following settings:

Back Buffer Format: Represents the data format individual pixels are stored as.
Depth/Stencil Format: Data format for z-buffer and stencil-buffer (unused).
Multisample Type: The quality of anti-aliasing. The higher the better the rendered images look like.

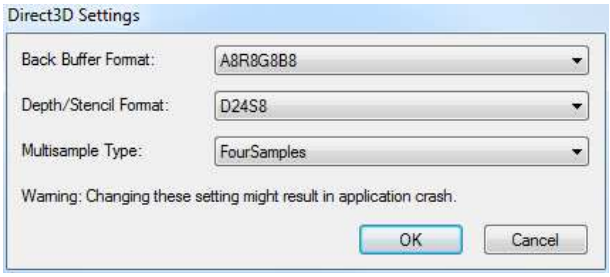


FIGURE 12.9. Direct3D settings.

7.2.10. *Rendering and Highlighting.* SiteBinder supports rendering of molecules using Microsoft Direct3D 9.

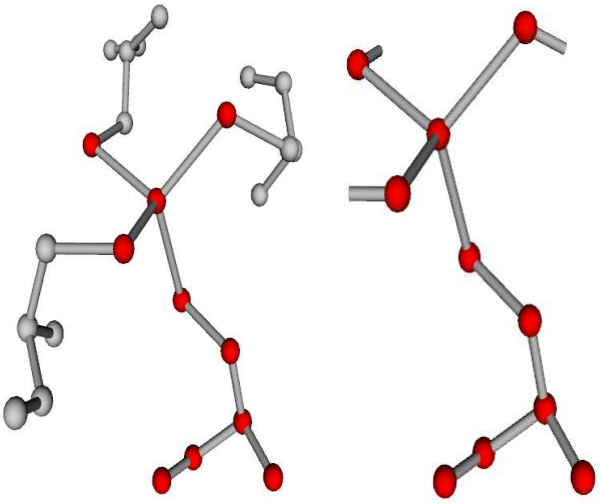
Display Types. Two display types are supported:

All: Displays all atoms of the structure (figure 12.10a).
Selected: Displays only selected atoms (figure 12.10b). Selected atoms are displayed using different color.

Highlighting (see below) ignores *display type*. Therefore, it is possible to see highlighted atoms even if they are not selected.

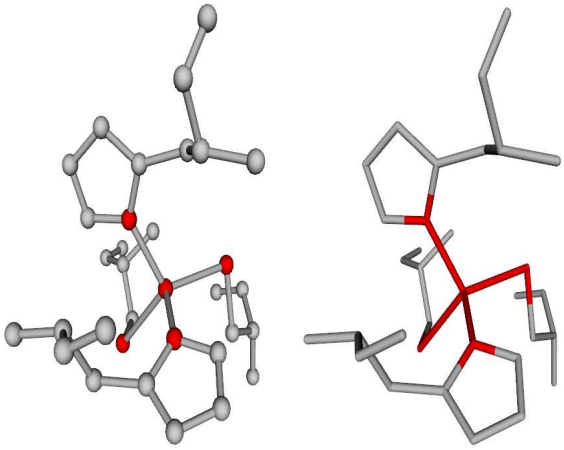
Display Modes. Two display modes are supported:

Balls: Atoms are displayed as balls (figure 12.11a).
Sticks: Only bonds are displayed (figure 12.11b).



(a) All. (b) Selected.

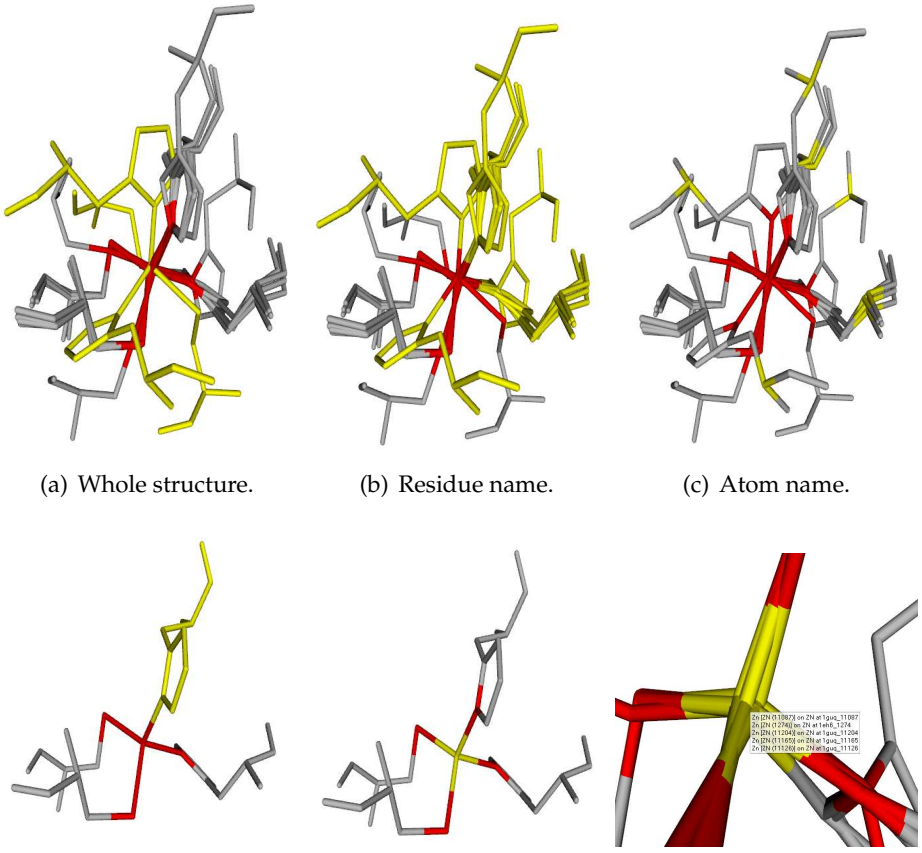
FIGURE 12.10. Display types.



(a) Balls. (b) Sticks.

FIGURE 12.11. Display modes.

Highlights. A variety of highlight types (figure 12.12) are supported by SiteBinder:



(a) Whole structure. (b) Residue name. (c) Atom name.

(d) Residue serial number. (e) Atom serial number. (f) Mouse-over.

FIGURE 12.12. Highlight types.

Whole structure: Whole structure is highlighted by hovering over its ID in the structure list. Furthermore, there is an option to select a persistent highlight of a structure that also distinguishes the given structure in the structure list.

Residues: All structures with a given set of residues are highlighted by hovering over the names of the residues in the Atom Selection Tree (multiple selection mode only).

Residue name: All residues with a given name are highlighted by hovering over the residue name in the Atom Selection Tree (multiple selection mode only).

Residue serial number: Residue with given serial number is highlighted by hovering over the residue in the Atom Selection Tree (single selection mode only).

Atom name: All atoms with a given name are highlighted. Only atoms on the original atom's residue name are considered by hovering over the atom name in the Atom Selection Tree (multiple selection mode only).

Atom serial number: Atom with a given serial number is highlighted by hovering over the atom in the Atom Selection Tree (single selection mode only).

Mouse-over highlight: If the number of displayed structures is not too large, hovering over atoms in the render window will give information about their element symbol, name, serial number, residue, residue identifier, and the structure identifier.

Coloring. Atoms can be colored based on whether they are selected or not. Both colors can be adjusted. Also, highlight color can be adjusted.

When a structure is loaded, it is assigned a random bond color to help distinguish individual structures. The bond color can also be adjusted.

Saving. Molecules can be rendered to a PNG image and saved to a file.

Five resolutions are supported: 640x480, 800x600, 1024x768, 1280x1024 and 1600x1200.

7.2.11. *Bond Lengths.* The values of bond lengths are stored in file `BondLengths.xml` and can be adjusted by the user. All values are in Ångströms.

Three types of lenghts can be adjusted:

Default Lenght: Specifies bond lenghts for "unknown" atoms. Can be adjusted by changing

```
<default value=x.y />
```

line in `BondLengths.xml`.

Atom Radius: Specifies "radius" of a given atom. Can be adjusted by changing/adding

```
<radius element=ElSymbol value=x.y />
```

line in `BondLengths.xml`.

Bond Length: Specifies the maximum distance of two atoms for them to contain a bond. Can be adjusted by changing/adding

```
<bond element1=ElSymbol element2=ElSymbol value=x.y />
```

line in `BondLengths.xml`.

CHAPTER 13

Design of SiteBinder

This chapter provides a brief overview of the internal architecture of SiteBinder.

1. Architecture

The source code is divided into three major parts:

Core: Contains the implementation of the algorithms presented in this thesis along with a structure manager for handling multiple structures at the same time.

Rendering: Contains the code for computing structure meshes and highlighting, and a control for rendering structures using Microsoft Direct3D.

UI: User interface.

SiteBinder.CMD is a stand-alone application and is just a simple interface to the SiteBinder Core.

1.1. Core. Figure 13.1 shows a class diagram of essential classes in SiteBinder Core.

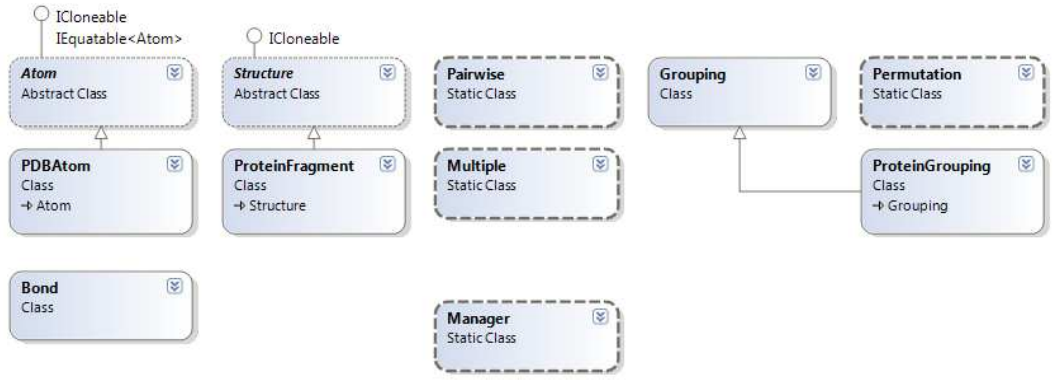


FIGURE 13.1. Class diagram of SiteBinder Core (contains selected classes only).

The implementation is as transparent as possible, allowing for easy extensibility of SiteBinder.

Representation of Structures. Structures are represented by the abstract class `Structure`. From this class individual types of structures are derived. Currently, only protein fragments loaded from the PDB file are supported by the class `ProteinFragment`

The class `Structure` contains the following fields:

Id: Identifier of the structure.

Bonds: List of bonds between atoms in the structure represented by the class `Bond`.

SelectedAtoms: List of selected atoms.

OtherAtoms: List of remaining atoms.

Atoms are represented by an abstract class `Atom` which contains the following fields:

Id: Identifier of the atom.

ElementSymbol: Element symbol of the atom, for example Zn.

Position: A three dimensional vector representing atom's position.

The class `PDBAtom` inherits from Atom and adds all the extra information available in PDB files (this class is used by the class `ProteinFragment` to represent atoms).

Math. SiteBinder Core provides an implementation of basic vector and matrix math is provided for 3D and 4D vectors, quaternions, and 3x3 and 4x4 matrices.

Also, the symmetric matrix eigenvalue decomposition algorithm from **[25]** has been integrated into the core.

Logging and Exceptions. SiteBinder provides logging information to better identify potential errors in the code and also allow seamless user experience.

The following logging items are supported:

Pairwise info: Detailed information about grouping, RMSD, and rotation is provided.
Multiple info: Detailed information about current actions of multiple superimposition algorithm.
Summary: Contains information about the last superimposition operation. Detailed information about error in the alignment if provided.

These exceptions are supported:

Initialization error: Details about an error while initializing SiteBinder Core.
Add error: If an error occurs while adding structures, the user is notified with a summary and can check the details in the log.
No atoms selected: Occurs when the user tries to superimpose structures with no atoms selected.
Selection compatibility: Occurs when two structures have different selected atoms.

Pairwise algorithm. The class `Pairwise` contains an implementation of the superimpose and best-fit algorithms for two structures.

The algorithm implementation is supported by classes `Grouping` (and consequently `ProteinGrouping` which implements the three grouping types suggested in chapter 6), `Permutation` (with `PermutationTable` for "small" permutations and the class `Permutations` for the general case – taken from **[37]**).

Multiple algorithm. The class `Multiple` contains an implementation of the superimposition algorithm for multiple structures.

Progress. SiteBinder Core supports the tracking of the progress of the current computation allowing the user to keep track of the computation. This functionality is embedded directly info classes `Pairwise` and `Multiple`.

Manager. A manager class is provided for better handling, adding, removing, selecting and superimposing multiple structures.

1.2. Rendering. The structure of SiteBinder Rendering is shown on figure 13.2.

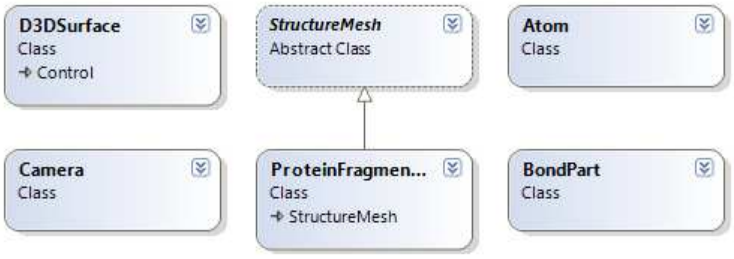


FIGURE 13.2. Class diagram of SiteBinder Rendering (contains selected classes only).

The class `D3DSurface` contains the user control for displaying graphics. The class `Camera` contains functions for manipulation with the camera allowing rotations and zooming of the rendered structures.

The class `StructureMesh` is a base class for rendering meshes of structures. From this class `ProteinFragmentMesh` is derived to allow support of highlighting based on information provided in PDB files (atom name, etc.).

The classes `Atom` and `BondPart` provide basic elements of structure meshes. Each atom contains a list of half-bonds (i.e. half of the mesh of the given bond) of the given atom.

1.3. User Interface. SiteBinder contains a WindowsForms [**38**] user interface. A class diagram 13.3 shows the classes of SiteBinder UI.

FIGURE 13.3. Class diagram of SiteBinder UI.

The main class of the user interface is `MainForm`, nevertheless, most functionality is implemented in the class `StructureListView` which contains a control for handling a list of structures including displaying selected atoms, residues and RMSD information. Description of other classes on the class diagram is straightforward.

Part 5

Results and Discussion

CHAPTER 14

Practical Applications

This chapter shows three practical applications of the SiteBinder software described in the previous chapters.

First, an example aligning multiple ZnS_4 binding sites is shown. Second, a model of Zinc finger [**39**] binding site is presented. Next, multiple lectin CaO_6 binding sites are superimposed showing very nice properties of this binding site. Finally, a summary of running times is provided.

All examples were generated on a standard desktop PC equipped with Intel Pentium Dual-Core (1.6 GHz), 4 GB RAM, and ATi Radeon 3850HD graphic card.

The input data were obtained by Bc. Petr Kovács as a part of his diploma thesis [**40**] and can be found on the enclosed disc.

1. ZnS_4 Binding Site

This introductory sample shows superimposition of 415 ZnS_4 binding sites (containing four CYS residues) with increasing number of atoms selected along with the corresponding models.

Figure 14.1a: Shows structures before being superimposed.

Figure 14.1b: Shows superimposition with the selection $\{Zn, 4xS\}$ and the corresponding model. RMSD = 0.123 Å was computed in $13.489\,s$ (6 iterations of the multiple superimposing algorithm were needed, see section 4 of chapter 7).

Figure 14.1c: Shows superimposition with the selection $\{Zn, 4xS, 4xC\}$ and the corresponding model. RMSD = 0.831 Å was computed in $25.630\,s$ (6 iterations needed).

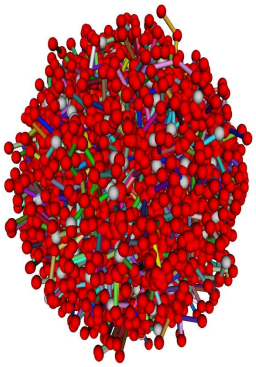
Figure 14.1d: Shows superimposition with the selection $\{Zn, 4xS, 8xC\}$ and the corresponding model. RMSD = 1.170 Å was computed in $30.988\,s$ (7 iterations needed).

Figure 14.1e: Shows superimposition with the selection $\{Zn, 4xS, 12xC\}$ and the corresponding model. RMSD = 1.449 Å was computed in $50.253\,s$ (8 iterations needed).

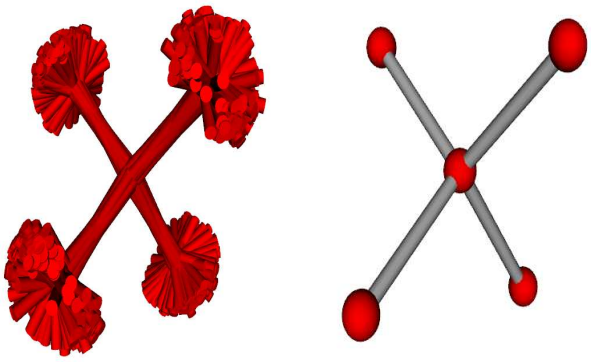
Figure 14.1f: Shows superimposition with the selection $\{Zn, 4xS, 12xC, 4xN\}$ and the corresponding model. RMSD = 1.657 Å was computed in $71.323\,s$ (7 iterations needed).

Figure 14.1g: Shows superimposition with the selection $\{Zn, 4xS, 12xC, 4xN, 4xO\}$ and the corresponding model. RMSD = 1.845 Å was computed in $71.693\,s$ (7 iterations needed).

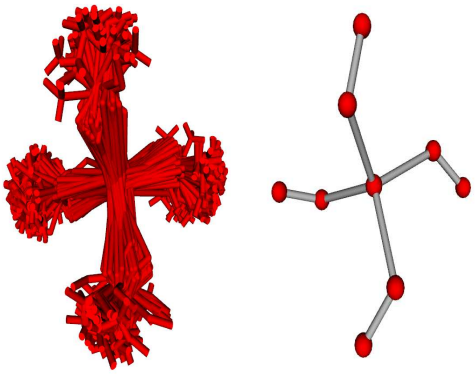
As we can see from the values of RMSD, each time the selection gets larger, the value of RMSD dramatically increases. Visually, this is in direct correspondence to the models being deformed in two branches, which gives us an information about the average spatial organization of these binding sites in proteins.



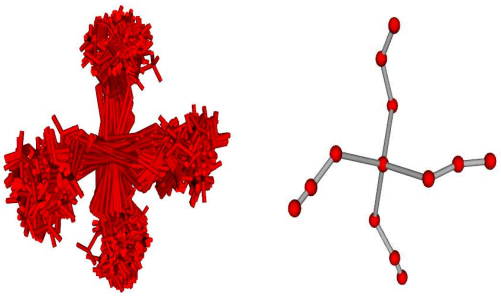
(a) Before superimposition.



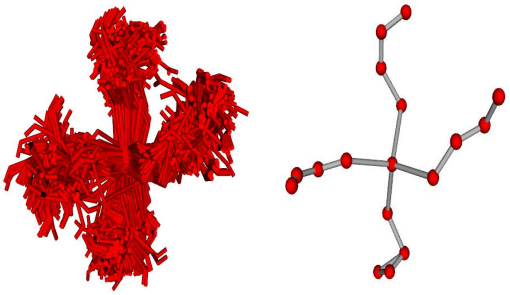
(b) 5 selected atoms (RMSD = 0.123 Å).



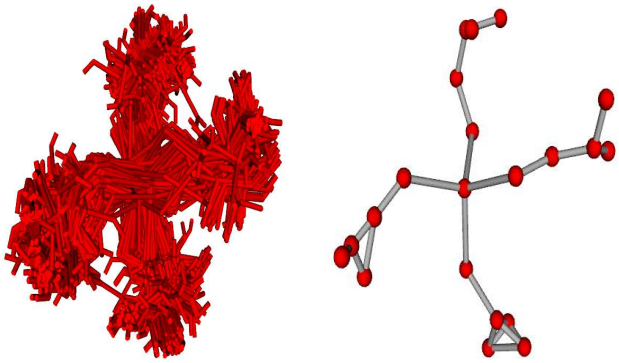
(c) 9 selected atoms (RMSD = 0.831 Å).



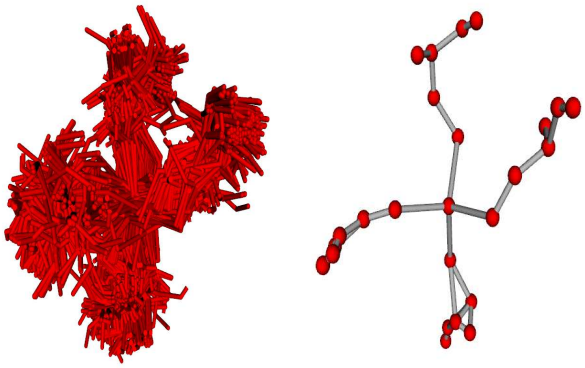
(d) 13 selected atoms (RMSD = 1.170 Å).



(e) 17 selected atoms (RMSD = 1.449 Å).



(f) 21 selected atoms (RMSD = 1.657 Å).



(g) 25 selected atoms (RMSD = 1.845 Å).

FIGURE 14.1. 415 ZnS_4 binding sites and the corresponding models.

2. Zinc Finger

Zinc fingers are small protein structural motifs that can coordinate one or more zinc ions to help stabilize protein folds. They can be classified into several different structural families and typically function as interaction modules that bind DNA, RNA, proteins or small molecules [**39**].

Figure 14.2 shows the superimposition and the model of 502 ZnN_2S_2 binding sites each containing two CYS and two HIS residues. The selection contains the atoms $\{Zn, 6xC, 4xN, 2xS\}$ and the result was obtained in $23.865\,s$ (4 iterations were needed). RMSD is 0.687 Å which indicates a good quality of the model.

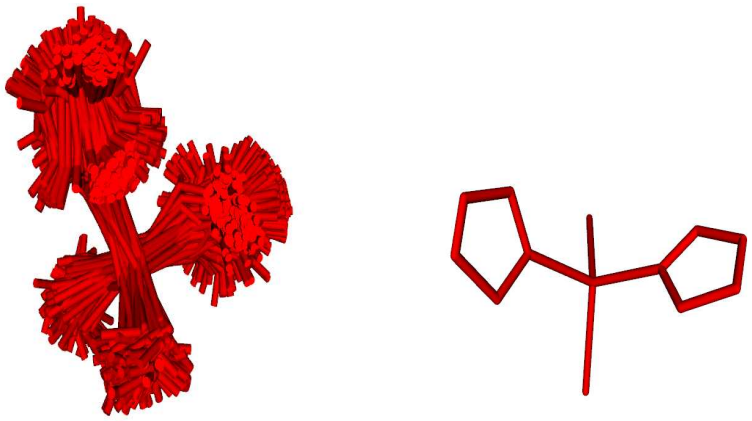


FIGURE 14.2. Superimposition of 502 ZnN_2S_2 binding sites (RMSD = 0.687 Å) and their model.

3. Lectin CaO_6 Binding Site

Lectins are sugar-binding proteins which are highly specific for their sugar moieties. They typically play a role in biological recognition phenomena involving cells and proteins. For example, some viruses use lectins to attach themselves to the cells of the host organism during infection [**41**].

Figure 14.3 shows superimposition of 16 lectin CaO_6 binding sites containing FUC, ASPx2, and ASNx2 residues. Each structure contains 43 selected atoms and the superimposition took $2.141\,s$ (4 iterations were needed). As we can deduce from the picture and the value of RMSD being 0.079 Å, these CaO_6 binding sites are remarkably similar, despite the fact that they were extracted from different proteins.

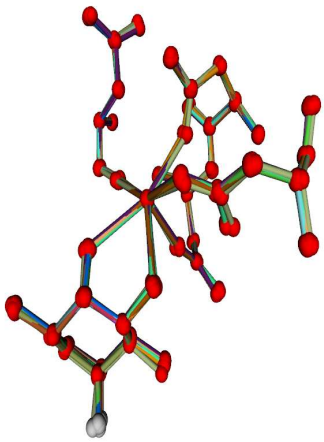


FIGURE 14.3. Superimposition of 16 lectin CaO_6 binding sites (RMSD = 0.079 Å).

4. Running Times

Running times were obtained on a standard desktop PC equipped with Intel Pentium Dual-Core (1.6GHz) and 4 GB RAM running Windows 7 operating system.

Table 14.1 contains a summary of running times and number of iterations needed before the multiple alignment algorithm converged for the examples presented in this chapter.

Binding Site	Structures	Atoms	Selection	Running Time	Iterations
ZnS_4	415	5	$\{Zn, 4xS\}$	$13.489\,s$	6
		9	$\{Zn, 4xS, 4xC\}$	$25.630\,s$	6
		13	$\{Zn, 4xS, 8xC\}$	$30.988\,s$	7
		17	$\{Zn, 4xS, 12xC\}$	$50.253\,s$	8
		21	$\{Zn, 4xS, 12xC, 4xN\}$	$71.323\,s$	7
		25	$\{Zn, 4xS, 12xC, 4xN, 4xO\}$	$71.693\,s$	7
ZnN_2S_2	502	13	$\{Zn, 6xC, 4xN, 2xS\}$	$23.865\,s$	4
CaO_6	16	43	$\{Ca, 22xC, 6xN, 14xO\}$	$2.141\,s$	4

TABLE 14.1. Running times for examples in this chapter.

Table 14.2 contains running times and numbers of iterations for increasing number of ZnN_2S_2 binding sites with selection $\{Zn, 6xC, 4xN, 2xS\}$ (13 atoms). *Running time 1* was obtained by using automatic grouping selection, whereas *Running time 2* shows the results when *element symbol* grouping was forced (see chapter 6 for details). From the results we can also compute the average time to superimpose a pair of structures: $12\,ms$[1].

Structures	Running time 1	Running time 2	Iterations
10	$0.460\,s$	$55.209\,s$	4
25	$1.366\,s$	$135.161\,s$	4
50	$3.366\,s$	$267.488\,s$	4
100	$3.112\,s$	$664.228\,s$	3
250	$10.112\,s$	N/A	4
500	$23.782\,s$	N/A	4

TABLE 14.2. Running times for increasing number of ZnN_2S_2 binding sites with 13 selected atoms. For *Running time 1* the default grouping is used and for *Running time 2* the *element symbol* grouping is forced.

We can see that the times are very reasonable even for larger number of structures when using the automatic grouping selection (which usually results in choosing *residue name* or *residue* grouping). This allows the user to "play" with the structures seeing immediate results.

In contrast, the times when *element symbol* grouping is forced are very long even for lower number of structures. This indicates that even though *residue name* and *residue* groupings still yield exponential complexity, it makes sense to use them over the "naive" approach. For example, for this particular case of ZnN_2S_2 binding site with two HIS and two CYS residues and selection $\{Zn, 6xC, 4xN, 2xS\}$, the pairing algorithm needs to test 68 cases when using the *residue name* grouping, but 34560 cases when the *element symbol* is used.

Finally, the numbers of iterations needed to superimpose the structures support the hypothesis about linear running time of the multiple superimposition algorithm (see section 5 of chapter 7).

[1]The value has been computed from the time to superimpose 500 structures: $23.782\,s/4/500 = 0.0118\,s$.

CHAPTER 15

Comparison with Other Software

SiteBinder was compared to three existing software packages that are able to superimpose structures - VMD, PyMOL, and Chimera. These programs are briefly described in chapter 11.

Only pairwise molecule superimposing parts (where available) of the software were compared. Multiple superimposition comparison was not concluded due to the technical difficulties of such a comparison (the problem is with varying definition of RMSD for multiple structures).

Two types of tests were concluded:

- RMSD of pairs of molecules have been compared.
- RMSD of one molecule has been compared with permutations of another molecule (order of atoms in the PDB file of the molecule was permuted).

The tests were concluded by Lukáš Pravda (as a part of his bachelor thesis [**42**]) and Jan Oppelt.

If no result is provided (where "-" is used instead of the numeric value), the software was unable to superimpose the structures.

1. Comparing Pairs of Molecules

Table 15.1 shows results obtained when comparing very similar structures. Tables 15.2 and 15.3 show results obtained when comparing structures that are not similar.

Structures	Atoms	SiteBinder	VMD	Chimera	PyMOL
1gzt_3548 1gzt_3574	44	0.065	0.065	0.065	0.054 (31) [1]
1w8f_3575 2jdh_3583	44	0.059	-	-	0.032 (29)
1w8h_3367 1w8h_3436	44	0.066	0.066	0.066	0.063 (30)
2jdk_3437 2jdk_3479	43	0.039	-	-	0.034 (32)
2vuc_3470 2vud_3395	43	0.139	1.600	1.600	0.121 (31)

TABLE 15.1. Comparison of CaO_6 binding site pairs. The unit is Ångström (Å).

Structures	Atoms	SiteBinder	VMD	Chimera	PyMOL
1a6y_2101 1a5t_3064	25	2.014	-	-	3.776 (24)
1bto_11314 1a71_5576	25	0.149	0.149	0.149	0.152 (24)
1a71_5628 1acm_7115	25	2.163	3.095	3.095	3.153 (24)
3h0r_63218 3gat_2091	25	1.398	-	-	3.339 (24)
1axe_5580 1a6y_2104	25	1.183	4.318	4.318	4.400 (24)

TABLE 15.2. Comparison of ZnS_4 binding site pairs. The unit is Ångström (Å).

[1] The number in parentheses indicates the number of atoms used for superimposition.

Structures	Atoms	SiteBinder	VMD	Chimera	PyMOL
1a1f_1148 1a1f_1149	33	0.318	0.318	0.318	0.298 (31)[1]
1llm_2013 1a1g_1146	33	0.384	0.384	0.384	0.303 (29)
2jwo_1254 5znf_501	55	3.047 / 3.539[2]	3.970	3.577	2.630 (26)
2emp_622 2emz_644	55	0.573	0.573	0.573	0.578 (54)
4znf_503 3imi_4679	33	2.282	-	-	2.307 (31)
2row_1377 2q1z_5190	33	3.109	-	-	-

TABLE 15.3. Comparison of ZnN_2S_2 binding site pairs. The unit is Ångström (Å).

2. Comparing Permutation of Single Molecule

In general, a PDB file can contain atoms in arbitrary order. That is why the following test was concluded.

Tables 15.4a, 15.4b, and 15.4c, show results obtained after superimposing two binding sites. The first molecule is fixed in all cases whereas the atoms in the second structure are permuted.

The permutations are achieved by first sorting the atoms based on information about their residue name. Afterwards, the residues are permuted and stored in separate files. Each of the tested structures contains 4 residues resulting in 24 permutations.

Both SiteBinder and PyMOL provided the same result in each case, therefore, the value is listed only once.

Figure 15.1 shows the ZnS_4 structure *4mt2_410* (green) superimposed to 24 permutations of structure *1kk1_3017*. All 24 permutations are correctly mapped to one position.

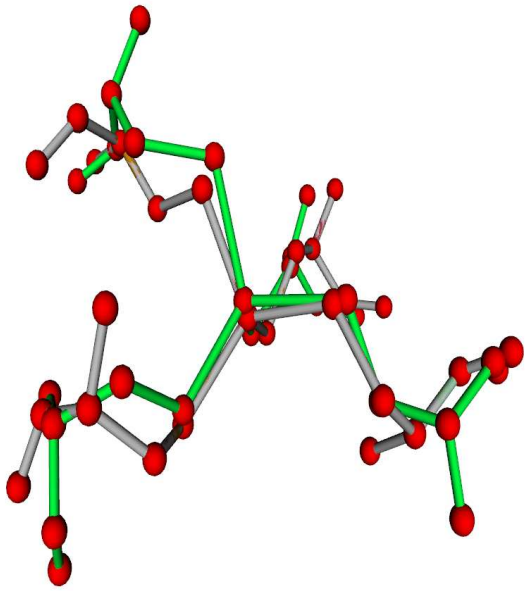


FIGURE 15.1. ZnS_4 binding site *4mt2_410* (green) and 24 permutations of *1kk1_3017*.

[1]The first result was obtained when reflection transformation was enabled (see chapter 5 for details).

[2]The number in parentheses indicates the number of atoms used for superimposition.

(a) ZnS_4: *4mt2_410* and permutations of *8atc_7127*.

SiteBinder	PyMOL	VMD
1.560	3.872	3.796
		1.729
		3.317
		2.832
		4.586
		3.428
		1.606
		3.902
		4.477
		3.574
		3.423
		2.883
		2.740
		3.405
		3.282
		4.671
		3.712
		1.643
		3.394
		4.520
		2.762
		3.480
		1.663
		3.706

(b) ZnS_4: *3hud_5565* and permutations of *6at1_7112*.

SiteBinder	PyMOL	VMD
2.096	3.234	3.175
		3.246
		3.960
		2.104
		2.670
		3.138
		3.465
		3.001
		2.907
		3.150
		3.781
		2.094
		2.446
		3.999
		3.496
		2.655
		3.060
		3.188
		3.471
		2.880
		2.335
		3.885
		3.418
		3.092

(c) ZnS_4: *5at1_7112* and permutations of *6at1_7112*.

SiteBinder	PyMOL	VMD
0.185	0.190	0.185
		3.577
		2.659
		3.735
		3.647
		4.526
		3.590
		1.180
		3.554
		4.897
		3.863
		3.636
		2.770
		4.403
		3.814
		1.079
		4.050
		4.892
		3.815
		3.815
		3.868
		2.857
		4.037
		1.448

TABLE 15.4. Comparing permutations of a molecule. The unit is Ångström (Å).

3. Discussion of the Results

With the exception of PyMOL, SiteBinder is at least as good as the other tested software for superimposing structures. The difference is bigger in favor of SiteBinder especially when the compared structures are more "different."

The better results sometimes provided by PyMOL are due to the fact that PyMOL uses a subset of atoms, rather than all selected atoms, to superimpose structures. Nevertheless, this feature is undesirable in our case, because it only gives us information about the similarity of structures and does not provide a suitable pairing of atoms which is needed for computing models of structures.

As is seen in the previous chapter (14), the running times for superimposing molecules of SiteBinder are very reasonable. This provides an interactive experience for the user who can easily "play" with the selections of atoms on individual structures and immediately see the results in *a click of a button*. It is in direct contrast with PyMOL and VMD which provide superimposing of structures only via scripting commands.

CHAPTER 16

Future Work

There are several directions to which the future development of this project can go. These can be split into two classes. The future work of the author of this thesis and the work that can be inspired by this thesis.

1. Future Work of the Author

The following list shows a possible development of future work on the topic of this thesis:

- Debug and improve SiteBinder. As in any larger computer software, there are bugs in SiteBinder waiting to be fixed and new ones to be reported.
- New features of the user interface can be added such as improvements of the atom selection interface or better support for large number of loaded structures.
- Improve support for PDB file format.
- Develop a version of SiteBinder user interface specialized for superimposing two structures only.
- Prove the upper bound of running time for multiple superimposition algorithm.
- Write two publications: The first about the version of SiteBinder mentioned above, followed by a publication about "full" version of SiteBinder.

Continue to work on this topic as a post-gradual student:

- There are many possible ways of improving the algorithm for finding the best pairing of atoms. These include the implementation of the pivot heuristics, or implementation of other types of grouping. Most notably, a support for grouping based on large protein complex structures would be a very desirable feature. There is also always the possibility of finding a new and more effective algorithm altogether. Actually, the algorithm was originally designed to superimpose only the binding atom and its closes neighbors and later somewhat artificially extended to support larger structures as well.
- A model of the binding site can be used as input for a possible algorithm for predicting binding sites in proteins.
- Extend the presented algorithms to arbitrary molecular structures. This might have applications for example in drug design.

2. Future Work Inspired by the Thesis

These are the ongoing or possible projects inspired by this thesis:

- The bachelor thesis of Lukáš Pravda **[42]** focused on comparison of software for superimposing molecular structures.
- Jan Oppelt's project comparing software for superimposing molecules.
- Writing a manual for SiteBinder.
- Developing a multi-platform user interface for SiteBinder. This task can be suitable for example for a bachelor thesis.
- Design and implement parallel version of the pairing algorithm. This is another good candidate for a bachelor (or master's) thesis. The parallelization should be straightforward (i.e. dividing the search space). The reason for parallelization is that modern processors contain many computing cores and therefore it would be possible to handle larger amount of structures in reasonable amount of time (i.e. short enough for the user to be able to "play" with the structures).

Part 6

Conclusion

CHAPTER 17

Conclusion

The main focus of this master's thesis is on superimposing multiple binding sites of proteins, and consequently creating their models. The author has succeeded in designing (or presenting) and implementing effective algorithms for superimposing small protein fragments, especially binding sites.

The algorithms presented in this thesis include superimposing two three dimensional structures, finding the pairing of atoms of two protein structures, and ultimately an algorithm for superimposing multiple structures at once.

The algorithm for superimposing two three dimensional structures is based on quaternion math and solving eigenvalue and eigenvector problems for 4x4 matrix in order to minimize value of RMSD (root mean square deviation). Assuming the input structures are not degenerate, a unique solution is provided.

The next algorithm finds the best pairing of atoms of two protein structures. The algorithm is based on sequential reordering the atoms in one of the structures and returning the pairing that gives the best value of RMSD obtained by the algorithm mentioned in the previous paragraph.

Finally, superimposing multiple structures at once is achieved by iteratively superimposing the structures to their average. Furthermore, when superimposing multiple protein binding sites, the final average structure serves as a model of that particular binding site.

The algorithms presented in this thesis were implemented in the software package called SiteBinder and C# was chosen as the implementation language for SiteBinder for its modern features such as the query language LINQ, and the portability of the code.

The input and output of the program is in PDB format, which was chosen for its rich capabilities of describing a protein structure. Furthermore, SiteBinder provides a user friendly graphical user interface (GUI) that allows the user a seamless manipulation with the loaded structures and comfortable selection of the atoms of interest. Moreover, the GUI provides real-time rendering capabilities that come along with another strong feature – the capability to highlight parts of the structures. All these features make SiteBinder a software that is easily accessible to an user without scripting/programming skills (which is often the drawback of other available software). Also, the program is quite effective and is capable of superimposing hundreds of binding sites in tenths of seconds. Last but not least, SiteBinder is a one of a kind software when it comes to superimposing a large number of small structures.

A simple command line interface to SiteBinder core was also implemented for users who wish to process their data in batches.

There are several ways to improve the presented algorithms and their implementation. For example, in some degenerate cases, a better value of RMSD could potentially be obtained. Nevertheless, SiteBinder was tested on real protein data and in most cases provided better or at least the same quality of results as established existing solutions such as VMD or PyMOL.

Finally, several practical applications of SiteBinder were presented.

In summary, effective, fast and reliable algorithms were designed, presented, and implemented in a unique software package called SiteBinder. SiteBinder was successfully tested on real protein data and it provides a basis for the development of new projects, such as predicting binding sites in proteins.

Bibliography

[1] Wikipedia. Active site — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Active_site.

[2] Wikipedia. Binding site — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Binding_site.

[3] Wikipedia. Protein — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Protein.

[4] Wikipedia. Metalloprotein — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Metalloprotein.

[5] Wikipedia. Root mean square deviation — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Root_mean_square_deviation.

[6] David Sehnal. Algorithms for comparing molecule conformations. Bachelor thesis, Masaryk University Brno, 2008.

[7] Wikipedia. Atom — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Atom.

[8] Wikipedia. Chemical bond — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Chemical_bond.

[9] Wikipedia. Molecule — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Molecule.

[10] Wikipedia. Amino acid — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Amino_acid.

[11] Wikipedia. Residue (chemistry) — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Residue_(chemistry).

[12] Wikipedia. Protein structure — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Protein_structure.

[13] E. W. Weisstein. Quaternion. MathWorld – A Wolfram Web Resource., 2009. [Online; accessed 25-December-2009]. Available from: http://mathworld.wolfram.com/Quaternion.html.

[14] Berthold K. P. Horn. Closed-form Solution of Absolute Orientation Using Unit Quaternions. *Journal of the Optical Society of America, volume 4*, 1986.

[15] Evangelos A. Coutsias, Chaok Seok, and Ken A. Dill. Using Quaternions to Calculate RMSD. 2004.

[16] Xueyi Wang and Jack Snoeyink. Defining and Computing Optimum RMSD for Gapped and Weighted Multiple-Structure Alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 5, no. 4.*, 2008.

[17] Wikipedia. Spectral theorem — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Spectral_theorem.

[18] E. W. Weisstein. Matrix trace. MathWorld – A Wolfram Web Resource., 2009. [Online; accessed 25-December-2009]. Available from: http://mathworld.wolfram.com/MatrixTrace.html.

[19] Coordinate File Description (PDB format). [Online; accessed 25-December-2009]. Available from: http://deposit.rcsb.org/adit/docs/pdb_atom_format.html.

[20] Wikipedia. Protein Data Bank — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Protein_Data_Bank.

[21] Wikipedia. .NET Framework — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/.NET_Framework.

[22] Wikipedia. Mono (software) — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Mono_(software).

[23] Wikipedia. Directx — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/DirectX.

[24] Wikipedia. Opengl — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/OpenGL.

[25] Sergey Bochkanov and Vladimir Bystritsky. ALGLIB.NET. [Online; accessed 25-December-2009]. Available from: http://www.alglib.net/.

[26] Sergey Bochkanov and Vladimir Bystritsky. Eigenvalues and eigenvectors of a real symmetric matrix. [Online; accessed 25-December-2009]. Available from: http://www.alglib.net/eigen/symmetric/symmevd.php.

[27] Wikipedia. LINPACK — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/LINPACK.
[28] Wikipedia. LAPACK — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/LAPACK.
[29] VMD — Visual Molecular Dynamics. [Online; accessed 25-December-2009]. Available from: http://www.ks.uiuc.edu/Research/vmd/.
[30] Wikipedia. PyMOL — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/PyMOL.
[31] UCSF Chimera. [Online; accessed 25-December-2009]. Available from: http://www.cgl.ucsf.edu/chimera/.
[32] QMOL. [Online; accessed 25-December-2009]. Available from: http://www.dnastar.com/qmol/.
[33] Swiss PDB Viewer. [Online; accessed 25-December-2009]. Available from: http://spdbv.vital-it.ch/.
[34] Gromacs. [Online; accessed 25-December-2009]. Available from: http://www.gromacs.org/.
[35] Wikipedia. C Sharp (programming language) — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/C_Sharp_(programming_language).
[36] Wikipedia. Language Integrated Query — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Language_Integrated_Query.
[37] Adrian Akison. Permutations, Combinations, and Variations using C# generics. [Online; accessed 25-December-2009]. Available from: http://www.codeproject.com/KB/recipes/Combinatorics.aspx.
[38] Wikipedia. Windows Forms — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Windows_Forms.
[39] Wikipedia. Zinc finger — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Zinc_finger.
[40] Petr Kovács. Binding Sites in Proteins. Master's thesis, Masaryk University Brno, 2010. Not finished while writing this thesis.
[41] Wikipedia. Lectin — Wikipedia, the free encyclopedia, 2009. [Online; accessed 25-December-2009]. Available from: http://en.wikipedia.org/wiki/Lectin.
[42] Lukáš Pravda. Comparing Software for Computing RMSD. Bachelor thesis, Masaryk University Brno, 2010. Not finished while writing this thesis.

Part 7

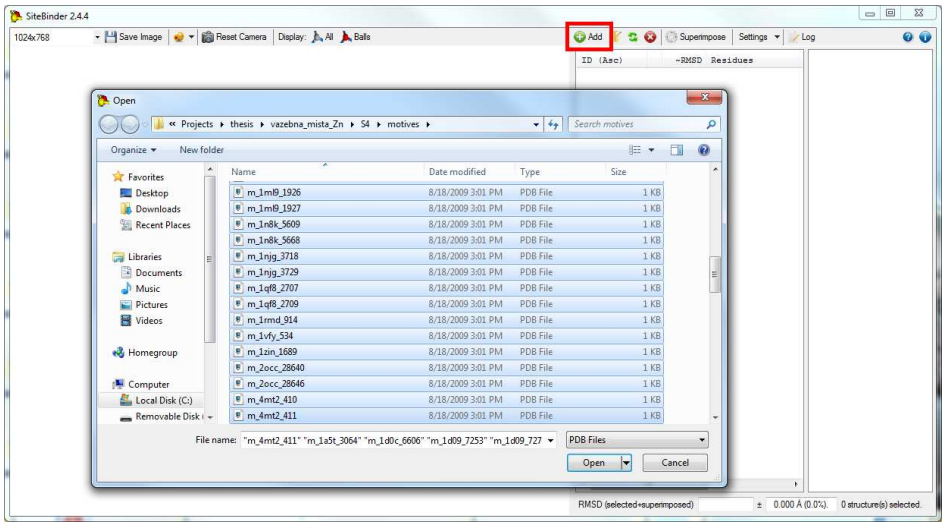
Appendix

APPENDIX A

Introduction to Using SiteBinder

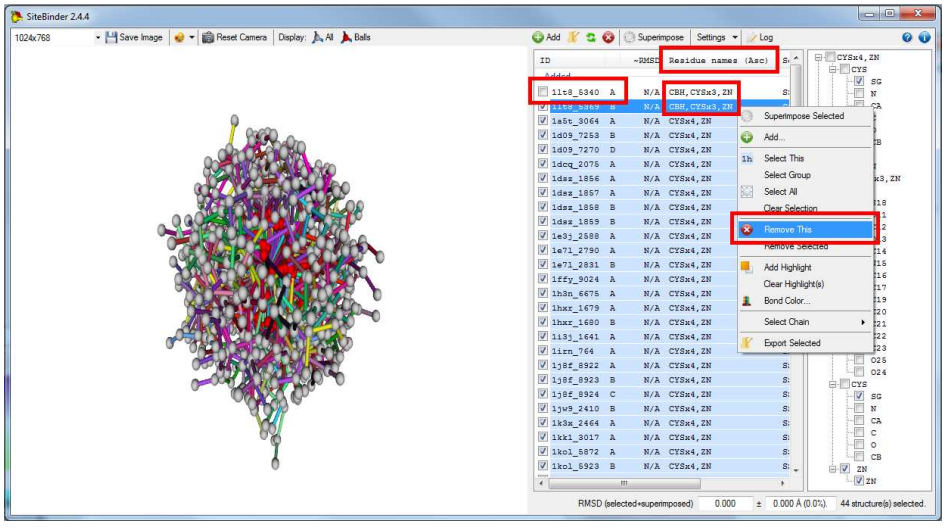
This chapter shows an introductory example of working with SiteBinder.

Loading Structures: To load structures click the "Add" button and a load dialog will pop-up. Alternatively, simply drag the PDB files into the structure list. The loaded structures are automatically selected.

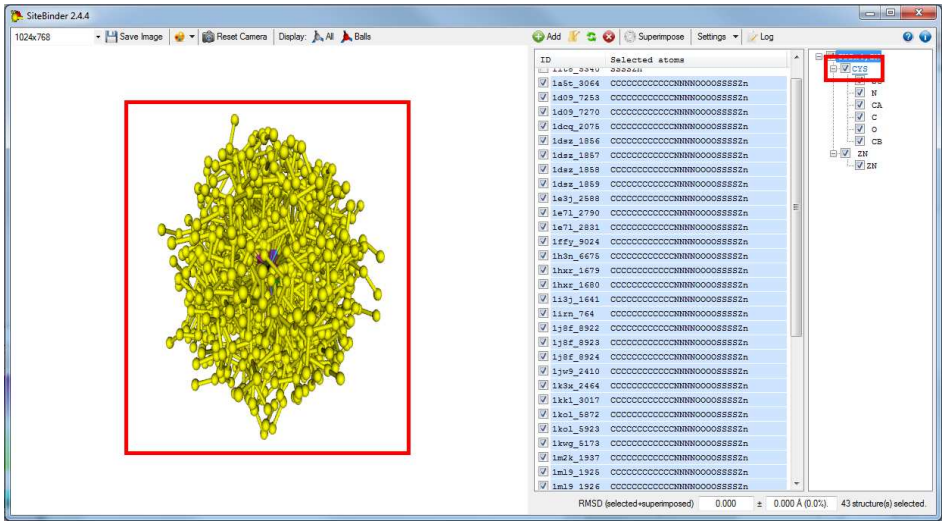


Selecting Structures of Interest: Let us suppose we want to superimpose all loaded structures containing four CYS residues. To achieve that, click "Residue names" header in the structure list and all loaded structures will get sorted lexicographically in ascending or descending order (depending on how many times the header was clicked).

Now, we can see that two of the loaded structures do not contain our desired four CYS residues. At this point, there are two options: by left-clicking the structure identifier the structure will de-select (multiple structures can be de-selected at once by dragging mouse over the desired identifiers), or by right clicking a structure and selecting "Remove this" option.

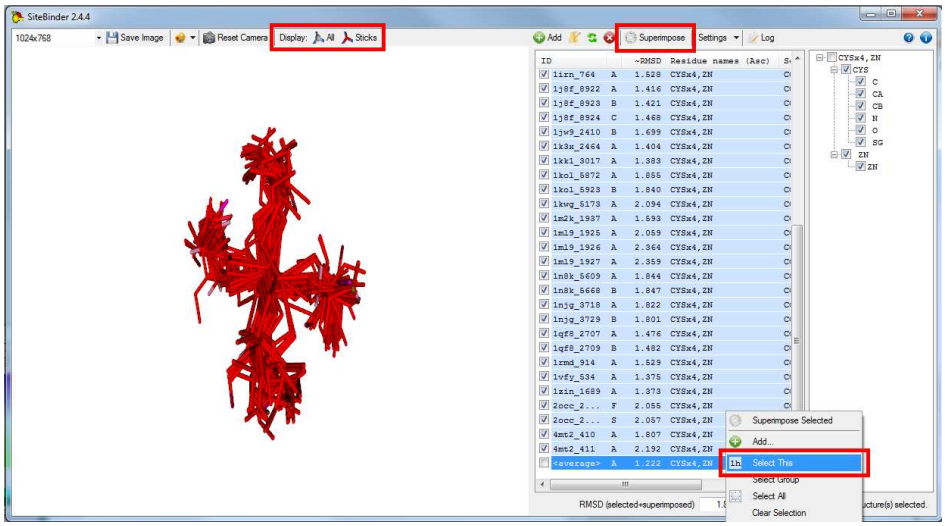


Selecting Atoms: Now that we have selected our structures of interest we are able to select all atoms on CYS residue by left-clicking "CYS" in the atom selection tree. Furthermore, once the mouse cursor is above the "CYS" text, all affected atoms are highlighted by yellow color.



Superimposing: Finally, click "Superimpose" button to superimpose selected structures. To adjust the display mode, click buttons next to "Display:" text.

To select only the average structure, right-click on "<average>" in the structure list and use "Select This" option.



Other Features: To access other features such as exporting, highlighting, etc., simply right-click on the structure list and choose the option of interest.

APPENDIX B

Contents of the Attached CD

- SiteBinder Visual Studio 2008 project is contained in the folder `Source`.
- Executable files are contained in the folder `Executable`.
- Microsoft .NET Framework 3.5 SP1 (folder `DotNetFramework`).
- Microsoft DirectX 9 (folder `DirectX`).
- Structures from part "Results and Discussion" (folder `Structures`).
- Text (in PDF format), LaTeX source code, and images of this thesis (folder `Thesis`).

List of Tables

List of Figures