Uppsala Master's Thesis in Computer Science
Examensarbete DV3
September 29, 2005

A System for Predicting Protein Function from Structure

Marta Łuksza

Information Technology
Department of Computer Systems

The Linnaeus Centre for Bioinformatics

Uppsala University
BOX 337
S-752 05 UPPSALA
SWEDEN

Supervisors: **Prof. Jan Komorowski**, **Dr. Torgeir R. Hvidsten**
The Linnaeus Centre for Bioinformatics, Uppsala University
Examiner: **Dr. David Ardell**
The Linnaeus Centre for Bioinformatics, Uppsala University

Abstract

Predicting protein function is a great challenge in molecular biology. More and more protein sequences are solved each year but the knowledge about their molecular role is still insufficient.

This thesis deals with the problem of predicting protein function from structure. Previously, Hvidsten et al [16] introduced a novel concept of popular multi-fragment local substructures of protein (local descriptors of protein structure) that are used for protein function prediction. In this thesis a software framework for computing, comparing and clustering descriptors was developed. The whole method results in a protein function classifier.

The problem involved dealing with very huge amount data. Various data mining techniques, such as clustering and learning classification rules, were employed. The very heavy computations were distributed and run on a computational grid.

A web based server was also developed, in order to facilitate applying the method to unknown, user-submitted proteins.

KEYWORDS: protein function prediction, protein structure, clustering algorithms, rough set theory

Acknowledgments

I would like to thank my supervisor Professor Jan Komorowski, I really appreciated the time spent at the Linnaeus Centre for Bioinformatics. I would also like to thank to Dr. Torgeir R. Hvidsten for guiding me in this project. I am very grateful to Dr. Witold Rudnicki and Marcin Kieczak for their wise advices. My friends Ewa Mąkosa and Jakub Jurkiewicz were always very helpful. Special thanks to my parents for their support.

Uppsala, September 29, 2005

MARTA ŁUKSZA

CONTENTS

1 INTRODUCTION

Proteins are the main building block of the cell. They are also responsible for its biological processes, for instance: gene regulating mechanisms, transportation of molecules, interaction with other molecules, and catalytic reactions that would be kinetically unfavorable.

The *central dogma of molecular biology*, formulated by Francis Crick in 1958, states that the genetic information contained in DNA is *transcribed* into RNA and subsequently *translated* into a protein. Genes, the coding parts of DNA, determine the sequence of amino acids that construct protein. The amino acids interact with each other and cause the protein to fold. Therefore the shape of a protein is specified by its amino acid sequence which itself is specified by genes. Moreover, the basic principle is that *function follows structure*, which means that the molecular role of a protein is determined by its shape.

There are many approaches to predicting protein function. For instance, function can be inferred from sequence - in this kind of approach the sequential similarity to other molecules is searched or sequence derived protein features are used [18]. However structure is much better evolutionary conserved than sequence and therefore it provides a better opportunity to recognize homology that is otherwise undetectable by simple sequence comparison. Only a small part of sequence, corresponding to the protein's active sites, needs to be conserved for keeping a function stable. The very first two protein structures solved, myoglobin and horse haemoglobin, were recognized as homologues, although their amino acid sequences did not reveal any significant similarity [5].

Sequence information is significantly more available than structure. Human genome, numerous microbial pathogens and model organism were sequenced while the structural information for the gene products is still insufficient. The *structural genomics* projects challenge the lack of structural information. Their main focus is on creating a representative set of solved protein structures to provide model structures for all the domains. Structures can be determined both experimentally (X-ray crystallography or NMR spectroscopy) and computationally.

Deriving function from structure is also the goal of structural genomics. Once a sufficient structural knowledge is gained, prediction of a reasonable structure and potential function for almost any protein will become possible. For instance, once a new protein structure is solved a structure comparison can be performed utilizing the Protein Data Bank [4], a database of known proteins. Two similar structures are very likely to have the same molecular role. However, this approach requires similarity of the whole molecule structures. The so-called fragment-based methods operating on small protein substructures are less demanding. The latter can be employed when neither sequence nor structure of significant similarity can be found for the whole investigated molecule.

1.1 Local descriptor based method

Local descriptor of protein structure, concept introduced in [23], is a small protein substructure encompassing short continuous backbone segments that are close in 3-dimensional

space but not necessarily along the protein sequence.

The local descriptors are employed in the protein structure prediction problem [17, 23] for identification of sequence signals that can be associated with those conformations. Hvidsten et al. [16] observe a correspondence between the presence of local descriptors in structure and protein function. The correspondence however is nonlinear, which means that usually a combination of local descriptors rather then one single descriptor can decide about the function.

In this thesis we developed and implemented a full system for predicting protein function based on the presence of some common descriptor conformations. The key idea of the method is to identify the most common descriptor shapes (by organizing them into clusters on a basis of their structural similarity) and then, using rough sets [27], associate those with protein functions. Figure 1.1 presents the process proposed, the process steps are as follows:

1. Calculating descriptors from the given set of proteins. All the structures are processed which results in a database of descriptors;
2. Descriptors from the database are compared to each other taking into account their 3-dimensional conformations. On a basis of those comparisons for each descriptor a group of similar descriptors is constructed;
3. On a basis of the grouping information, the descriptors are combined into non overlapping clusters of objects revealing more similarity to each other than to the objects from the other clusters. This step results in a database of clusters and cluster representative descriptors;
4. In the last process step a rule based classifier is constructed for predicting protein function on the grounds of the common conformations adopted by the protein. The Gene Ontology [2] annotations are used as decision classes while learning the classifier. The rules obtained are of the IF-THEN form, associating the presence of particular local descriptors with particular protein functions.

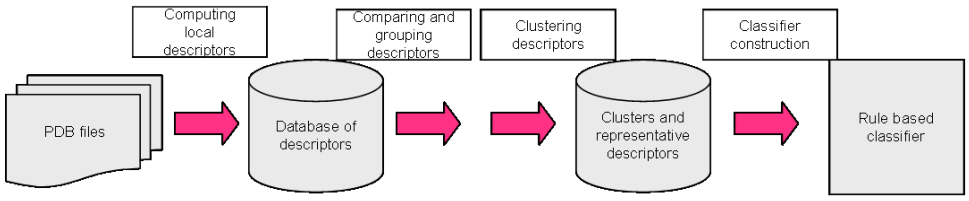

Figure 1.1. Local descriptors based protein function prediction

The method requires a large set of representative proteins to be processed, the subsequent steps also require dealing with huge amount of data. The method presents a typical data mining problem, where the knowledge is induced from data and various techniques are used to find hidden associations and characteristics. Formally, data mining can be defined as a *nontrivial extraction of implicit, previously unknown and potentially useful information from data* [11]. The valuable information is relatively hard to detect, however interpretable, and it is still true when new cases are observed. The well known data mining techniques applied in this thesis are clustering and learning classifications rules. However the whole method is a specific knowledge discovery process.

1.2 Objectives

The implementation goal of this thesis was a software framework automating the local descriptor method. This involves implementation of the steps of calculating, comparing and clustering descriptors (steps 1, 2 and 3). In the last step, where the classifier is constructed, the Rosetta system [22] is used.

The main challenge was to design a system capable of processing huge datasets, taking into account both the storage and the computational time. Some parts of the process had to be parallelized and run on a computer grid, the results of all of the process stages had to be stored in a database.

The other objective was to come up with a suitable clustering algorithm. Two algorithms were proposed, a density based DBSCAN algorithm [10] and a hierarchical algorithm [15]. The classification step depends strongly on the cluster assignments, therefore the algorithms are compared accordingly to the classifier performance. Furthermore, the cluster assignments quality was evaluated using the mutual information criterion, proposed in [13].

To enable application of the method by a wide range of users, a web server with a web user interface was developed. For each protein submitted, the rule based classifier is applied and the predicted function is returned.

1.3 Overview

The first part of the thesis presents a mathematical background of the problem. In Chapter 2 a review of clustering methods and an introduction to the rough set theory, used for classification, are given, covering notions needed for understanding of following chapters. Chapter 3 provides a description of the data, algorithms and methods constituting the process shown in Figure 1.1. In Chapter 4 the implementation aspects are discussed, such as the system architecture, adoption of the grid resources and the web server realization. Chapter 5 discusses the descriptor data and reports the results of clustering and classifier construction. Finally, the discussion summarizing the project is provided in Chapter 6. Appendix chapter provides the notation abbreviations used in the thesis and diagrams concerning the implementation aspects.

2 THEORETICAL BACKGROUND

This chapter provides an overview of the applied mathematical theory. Section 2.1 is an overview of clustering algorithms that belong to the unsupervised learning methods in machine learning. In Section 2.2 introduction to the rough set theory is given, which is a supervised method for learning from training examples. The notion of a local descriptor, Gene Ontology and other related concepts are described in following chapters.

2.1 Clustering

The goal of cluster analysis is to group data into subsets, so that objects within a cluster are more closely related to each other than those that belong to different clusters. As side effects one can achieve selection of the representative objects within the "homogamic" groups, discovery of new features or noise in the data. In this approach the cluster analysis methods will be used to find the most common local descriptors shapes and to reduce the space of all the descriptors to the set of the cluster-representative descriptors. Removing the least frequent shapes would be also desirable.

2.1.1 Data representation

An object from a dataset D, which is to be clustered, can be described by a set of attributes or by its relation (proximity) to other objects. The latter is the case of the local descriptor data, for each pair of descriptors their similarity is assessed.

The *dissimilarity matrices* represent the data as an $N \times N$ matrix $Diss$, where N is the number of objects from the dataset and d_{ij} matrix entry records dissimilarity between the i^{th} and the j^{th} objects. The dissimilarity matrix is often, by many clustering algorithms, assumed to be symmetric. If the original input data matrix does not satisfy this demand, it might be substituted by a symmetric matrix $(Diss + Diss^T)/2$.

The other way of representing data is by *dissimilarities based on attributes*. Very often data is represented as a matrix X of size $N \times M$, where number of objects is N and there are M attributes describing the object. Attributes can be regarded as functions $A_i : D \to V_i$, where D is a dataset to be clustered and V_i is a set of possible values for the i^{th} attribute. This kind of data representation can be easily transformed into dissimilarity matrix representation, a dissimilarity functions for each attribute has to be defined and then the whole dissimilarity is computed as a sum over these dissimilarities:

$$Diss(a, b) = \sum_{i=1}^{M} d_i(a_i, b_i),$$

where $d_i : D \times D \to \mathbb{R}$. The common choice for attribute dissimilarities function is the Euclidean distance.

2.1.2 Clustering algorithms

There are three basic types of clustering algorithms: partitioning, hierarchical and density based. In this section the brief characteristics for each type are covered, the algorithms applied are discussed in detail in Chapter 3.

Partitioning algorithms

Partitioning algorithms attempt to split a dataset into K (where K is an input parameter) clusters so that the partition optimizes a given objective function. These algorithms usually start with an initial partition, and then perform a series of iterative steps. At each step the cluster assignments are changed so that the value of the objective function is improved. The algorithm terminates when changing cluster assignments leads to no improvement and the current assignment is returned as a solution.

The algorithm solving the more general case without the limit on the clusters number, known as the *minimum sum-of squares clustering*, is NP-hard [7]. Hence the algorithms that are commonly used are heuristic, i.e. they search for the optimal value of the objective function but they do not guarantee to find a globally optimal solution.

K-means algorithm is one of the most popular partitioning algorithms. For each of the K clusters a *centroid* point - a gravity centre is computed. Next the algorithm tries to assign points to clusters so that the mean square distance of points to the assigned centroid cluster is minimized. This is repeated until the changing of centroids does not change the cluster assignments or the given stop condition is satisfied. In each such step the distances between each of K centres and the dataset points are computed, therefore it requires $O(K * n)$ operations. The number of iterations is nondeterministic. The centroid-based algorithms are suitable only for data in metric spaces, such as Euclidean space, where it is possible to compute a centroid point.

K-medoids algorithm works for data in similarity space, that is with an arbitrarily defined dissimilarity matrix $Diss$. It tries to find clusters representative points (*medoids*) so that the distances of points to their closest medoid are minimized:

$$c_k^* = argmin_{c \in C_k} \sum_{c' \in C_k} Diss(c, c')$$

where c_k is a medoid for cluster C_k. Searching the new cluster medoid requires $O(|C_k|^2)$ operations, whereas the corresponding operation in K-means was of linear cost. If all the clusters are of similar size, then we can estimate one iteration cost as $O(K * (\frac{N}{K})^2) = O(\frac{N2}{K})$ which is $O(N^2)$ for big datasets or small K. Therefore the K-medoids algorithm is much more computationally intensive than K-means. The well known approximate realizations of the K-medoid algorithm are PAM (Partition around medoids) [20], CLARA (Clustering Large Applications) [20] and CLARANS (Clustering Large Applications based on Randomized Search) [26].

Partitioning algorithms results are nondeterministic: they depend strongly on the choice of the initial partition and the parameter K. A drawback of both the centroid and medoid-based approaches is that they are not suitable for data in which points in a given cluster are closer to the center of another cluster than to the center of their own cluster. Such

situation might appear when sizes of clusters vary much or when cluster shapes are not convex.

Hierarchical algorithms

Hierarchical algorithms decompose a dataset D of N objects into several levels of nested clusters that can be represented by a dendrogram, a binary tree where the parent is a cluster constructed by merging its two daughter clusters. In order to get the cluster assignments, the tree is cut at a specified level, and the obtained subtrees represent the clusters. There are two basic strategies for hierarchical clustering: *agglomerative* (bottom-up) and *divisive* (top-down).

Agglomerative algorithms start at the bottom, with singleton clusters for each data point. In the consecutive, iterative steps, the selected pair of the two closest clusters is merged into one cluster at the next level. Let C_1 and C_2 be two clusters. The dissimilarity between them is computed from the pairwise objects dissimilarities belonging to those clusters. Define a dissimilarity function for clusters $DissClust : P(D) \times P(D) \rightarrow R \cup \{\infty\}$, where $P(D)$ is a set of all subsets of D. The clusters dissimilarities are computed from the set of pairwise dissimilarities. There are three commonly used measures for evaluating clusters dissimilarities [15]:

- *single linkage* $DissClust(C_1, C_2) = min_{c_1 \in C_1, c_2 \in C_2}(Diss(c_1, c_2))$
- *complete linkage* $DissClust(C_1, C_2) = max_{c_1 \in C_1, c_2 \in C_2}(Diss(c_1, c_2))$
- *average linkage* $DissClust(C_1, C_2) = avg_{c_1 \in C_1, c_2 \in C_2}(Diss(c_1, c_2))$
  $= \frac{1}{|C_1||C_2|} \sum_{c_1 \in C_1} \sum_{c_2 \in C_2} Diss(c_1, c_2)$

If we define a cluster diameter [15] as $D_C = max_{c_1 \in C, c_2 \in C} Diss(c_1, c_2)$, then we might expect to get very large diameters in case of the single linkage. This feature will allow clusters to be elongated and not necessarily spherical. To the contrary, the diameter will be quite small for complete linkage, where clusters are always convex. In contrast to the partitioning algorithms, hierarchical clustering is deterministic. For given settings it always results in the same solution.

Divisive algorithms start at the top, with one cluster containing all the points. At each level one of the existing clusters is split into two daughter clusters. A partitioning algorithm can be used to split the parent cluster, for instance K-means or K-medoids algorithm with the K parameter set to 2. There are many ways of choosing the cluster to be split, for instance the cluster with the biggest diameter, or the one that has the largest pairwise members dissimilarity.

Density-based algorithms

Density-based clustering algorithms apply a local cluster criterion. Clusters are regarded as regions in the data space in which the objects are dense, and which are separated by regions of low object density, called *noise*. Clusters can have an arbitrary shape and the

objects inside a cluster region may be arbitrarily distributed. The most popular density-based algorithm is probably DBSCAN [10]. Other known algorithms are: GDBSCAN (Generalized Density-Based Spatial Clustering of Applications with Noise) [30], OPTICS (Ordering Points To Identify the Clustering Structure) [1], and LOF (Local Outlier Factors) [6].

2.2 Supervised learning

Supervised learning [15] belongs to the fields of artificial intelligence, data mining and statistics. In the typical learning problem an outcome measurement, often called a *decision attribute* is given. The aim is to develop a model, a *classifier*, that will allow for predicting the outcome on a basis of a set of features. The term "supervised" corresponds to the presence of the decision attribute, which guides the process of learning. The decision attribute can be either *quantitative* (when the measurements can be compared to each other) or *qualitative*, like the Gene Ontology annotations in this setting. To construct a predictive model, a *training set* is used in which both the features and the decision attribute are provided. In the local descriptor problem, objects are proteins, features correspond to the possession of certain local descriptor, and the outcome corresponds to the protein function. There are various supervised learning algorithms known, such as the Nearest Neighbor method, linear regression, Bayesian methods [12] or decision trees [11, 15]. Here, a *rough set* approach is applied, which is discussed in more detail in the following section.

2.2.1 Rough set theory

Rough set theory, introduced by Zdzisław Pawlak [27], is an extension of set theory used to approximate sets that cannot be precisely described using the knowledge available. In here, the notion of training data is formalized.

Definition 2.2.1. [27] An *information system* is a pair $\mathbb{A} = (U, A)$, where U (*universe*) is a non-empty, finite set of examples, and $A = \langle a_1, ..., a_n \rangle$ is a non-empty, finite set of attributes. Each attribute a is a partial function of a form $a : U \rightarrow V_a$, where V_a is a *value set* of a, containing all the possible values of attribute a. Each object u from the universe U is a vector $\langle a_1(u), ..., a_n(u) \rangle$.

The observations, i.e objects from the set U, correspond to the observed configurations of feature values, and they approximate the reality description. The description reveals the hypothetical laws which we would want to formalize having incomplete information.

Definition 2.2.2. A *decision table* is an information system $\mathbb{A} = (U, A \cup \{d\})$,where $d \notin A$ is a distinguished *decision attribute*. The elements of A are called *conditions*.

An example of a decision table $(U, A \cup \{function\})$ is shown in Table 2.2.1. In here, the attributes set is $A = \{$1pjza_#132, 1jmxb_#56, 1ij5a_#72, 1bo9a_#12, 1qqva_#34 $\}$. The attributes correspond to representative descriptors (see Section 3.2.1 for an explanation of the descriptor naming convention). The decision attribute is $function$. The observations correspond to proteins, the value set for each of the attributes is $\{0, 1\}$, where 1 stands for the presence of the specific descriptor in the protein and 0 stands for its absence.

Table 2.1. Decision table

	$1pjza_{\#}132$	$1jmxb_{\#}56$	$1ij5a_{\#}72$	$1bo9a_{\#}12$	$1qqva_{\#}34$	*function*
1n9l	1	1	0	0	0	GO:0005524
1a33	1	1	0	1	0	GO:0005524
1b45	1	1	0	0	0	GO:0005524
1zme	1	1	0	1	0	GO:0008270
1afe	0	0	0	0	0	GO:0008270
1bkc	1	0	0	0	1	GO:0004222
1fk2	1	0	0	1	1	GO:0004222
1hk1	1	0	0	0	1	GO:0004222
1ufk	0	1	1	1	1	GO:0004222
1ufk	0	1	1	1	1	GO:0008757
2tps	0	1	1	0	1	GO:0016765

Definition 2.2.3. Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision system. Every $v_d \in V_d$ partitions the universe U in $|V_d|$ classes $X_1, ..., X_k$. Each class X_j, $(j \in \{1, ..., |V_{d_i}|\})$ is called a *decision class*.

Reducts

Some objects in a decision table may have the same values for the conditional attributes and belong to different decision classes. For example, proteins 1a33 and 1zme from Table 2.2.1 cannot be discerned based on the conditional attributes. These two proteins have similar structures but different functions. If a protein has more than one function known, there is an object in a decision table for each protein-annotation pair. Those objects refer to the same protein, therefore they have the same attribute values while belonging to different decision classes.

Definition 2.2.4. Let $\mathcal{A} = (U, A)$ be an information system and let $B \subseteq A$. A *B-indiscernibility* relation is an equivalence relation

$$IND_{\mathbb{A}}(B) = \{(u, u') : U \times U : \forall a \in B, a(u) = a(u')\}$$

Relation $IND_{\mathbb{A}}(B)$ is symmetric, reflexive and transitive hence it is an equivalence relation. The equivalence classes obtained from $IND_{\mathbb{A}}(B)$ are denoted $[x]_B, x \in U$, i.e. the B-indiscernibility relation groups the observations having the same values for the attributes from the set B. In other words, all the observations belonging to the same equivalence class with respect to B are *indiscernible* with respect to attributes from B.

Definition 2.2.5. Let $\mathbb{A} = (U, A)$ be an information system and let $B \subseteq A$. Set B *defines* set A if

$$IND_{\mathbb{A}}(B) = IND_{\mathbb{A}}(A) \tag{2.1}$$

B is a *reduct* when (2.1) is satisfied for B but it is not satisfied for any subset of B.

For example the set $\{1pjza_{\#}132, 1jmxb_{\#}56, 1bo9a_{\#}12\}$ is a reduct because:

$$IND_{\mathbb{A}}(\{1pjza_{\#}132, 1jmxb_{\#}56, 1bo9a_{\#}12\}) = IND_{\mathbb{A}}(A)$$

Decision rules

Based on reducts, the decision rules can be constructed. All the following definitions lead to the formal definition of a decision rule.

Definition 2.2.6. Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision table. An *atomic descriptor* is a pair (a, v) where $a \in A$ and $v \in V_a$.

Definition 2.2.7. Let $\mathbb{A} = (U, A)$ be an information system and $B \subseteq A$. *Language* τ_B is the smallest family containing all the descriptors (b, v), $b \in B$, $v \in V_b$, closed with respect to *conjunction*, i.e.:

If $\alpha \in \tau_B, \beta \in \tau_B$ then $\alpha \wedge \beta \in \tau_B$.

Analogically, *language* ξ_B is the smallest family containing all the descriptors (b, v), $b \in B$, $v \in V_b$, closed with respect to *alternative*, i.e.:

If $\alpha \in \xi_B, \beta \in \xi_B$ then $\alpha \vee \beta \in \xi_B$.

Definition 2.2.8. [32] For each formula $\alpha \in \tau_B$ there corresponds a *support* $\|\alpha\|_{\mathbb{A}} \subseteq U$, defined recursively:

- if α is an atomic descriptor (b, v): $\|\alpha\|_{\mathbb{A}} = \{u \in U : b(u) = v\}$
- if α is a formula $\beta \wedge \gamma$: $\|\alpha\|_{\mathbb{A}} = \|\beta\|_{\mathbb{A}} \cap \|\gamma\|_{\mathbb{A}}$

Analogically, for each formula $\alpha \in \xi_B$ there corresponds a *support* $\|\alpha\|_{\mathbb{A}} \subseteq U$, also defined recursively:

- if α is an atomic descriptor (a, v): $\|\alpha\|_{\mathbb{A}} = \{u \in U : b(u) = v\}$
- if α is a formula $\beta \vee \gamma$: $\|\alpha\|_{\mathbb{A}} = \|\beta\|_{\mathbb{A}} \cup \|\gamma\|_{\mathbb{A}}$

Definition 2.2.9. Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision table, τ_A be the language closed with regards to conjunction, and $\xi_{\{d\}}$ language closed with regards to alternative, both defined in 2.2.7. A *decision rule* for the decision table is an expression of the form $\varphi \Rightarrow \psi$, where $\varphi \in \tau_A$, $\psi \in \xi_{\{d\}}$. φ is the rule *antecedent* and ψ is the rule *consequent*.

For example the following rules can be obtained for reduct $B = \{$1pjza_#132, 1jmxb_#56, 1bo9a_#12$\}$:

1pjza_#132(1) $\wedge$ 1jmxb_#56(1) $\wedge$ 1bo9a_#12(0) $\Rightarrow$ function(GO:0005524)

1pjza_#132(1) $\wedge$ 1jmxb_#56(1) $\wedge$ 1bo9a_#12(1) $\Rightarrow$ function(GO:0005524) $\vee$ function(GO:0008270)

1pjza_#132(0) $\wedge$ 1jmxb_#56(1) $\wedge$ 1bo9a_#12(1) $\Rightarrow$ function(GO:0004222) $\vee$ function(GO:0008757)

1pjza_#132(0) $\wedge$ 1jmxb_#56(1) $\wedge$ 1bo9a_#12(0) $\Rightarrow$ function(GO:0016765)

1pjza_#132(1) $\wedge$ 1jmxb_#56(0) $\Rightarrow$ function(GO:0004222)

There are several quantities to describe a decision rule in a decision table:

Definition 2.2.10. A decision rule $\varphi \Rightarrow \psi$ *matches* an object $u \in U \Leftrightarrow u$ supports (see Definition 2.2.8) φ, i.e. $u \in \|\varphi\|_{\mathbb{A}}$. The number of the objects matching a decision rule $\varphi \Rightarrow \psi$ is denoted as $Match_{\mathbb{A}}(\varphi \Rightarrow \psi)$, $Match_{\mathbb{A}}(\varphi \Rightarrow \psi) = |\|\varphi\|_{\mathbb{A}}|$

Definition 2.2.11. The *support* of a decision rule $\varphi \Rightarrow \psi$, $Supp(\varphi \Rightarrow \psi)$ is the number of objects $u \in U$ that support both the antecedent and the consequent of the rule, i.e. $|\|\varphi\|_{\mathbb{A}} \cap \|\psi\|_{\mathbb{A}}|$, where $\varphi \in \tau_A$ and $\psi \in \xi_{\{d\}}$.

Definition 2.2.12. The *coverage* of a decision rule $\varphi \Rightarrow \psi$ is the fraction of the number of the objects that support the rule and the number of objects that support its consequent ψ, i.e. $\frac{Supp(\varphi \Rightarrow \psi)}{|\|\psi\|_{\mathbb{A}}|}$, where $\psi \in \xi_{\{d\}}$.

Definition 2.2.13. The *accuracy* of a decision rule $\varphi \Rightarrow \psi$ is defined as

$$Accuracy(\varphi \Rightarrow \psi) = \frac{Support_{\mathbb{A}}(\varphi \Rightarrow \psi)}{Match_{\mathbb{A}}(\varphi \Rightarrow \psi)}$$

2.2.2 Classifier

Finally, a classifier can be formally defined as:

Definition 2.2.14. A *classifier* $\hat{d}$ over an information system $\mathbb{A} = (U, A)$ is a function $\hat{d} : U \to V_d$ that maps an object $u \in U$ to the value of the decision attribute d from a decision table $\mathbb{A}_{\mathbb{T}} = (U_T, A \cup \{d\})$, where $U_T \subseteq U$ is a training dataset used for inducing the rules. Value of $\hat{d}(u)$ is a *prediction* of a decision value for u.

Classifier correctness

The intention is that the classifier should perform well on unseen objects, not present in the training set. A simple statistic for evaluating the classifier quality is *accuracy*, the ratio of the correctly classified objects. Usually the classifier accuracy is assessed from its performance on a *test set*, that the classifier was not trained on. In case of small datasets *cross-validation* is commonly used. The dataset is divided into K disjunctive parts (folds) of equal size. There are K iterations performed, in the i^{th} step the i^{th} fold is used as a test set, and the other objects are used for training the classifier. The accuracy is assessed as a mean accuracy over the K iteration steps.

If we are interested in how well the classifier predicts certain classes, other statistics have to be used. The following values can be computed with regards to a particular decision class:

- TP - true positives, the number of objects that were correctly classified to the decision class,
- TN - true negatives, the number of objects that do not belong to the decision class and were not classified to it,
- FP - false positives, the number of object that were incorrectly classified to the decision class,

- FN - false negatives, the number of objects that do belong to the decision class but were incorrectly classified to some other class.

Sensitivity is the fraction of correctly classified instances from the class, $\frac{TP}{TP+FN}$. *Specificity* is the fraction of the instances that did not belong to the class and were not classified to it, $\frac{TN}{TN+FP}$.

When applying a rule based classifier an object may be matched by several rules which cast votes for some classes. A threshold for the number of votes can be specified in order to assign the object to the class. Next, by plotting the $(1 - sensitivity)$ against $specificity$ for various values of the threshold we obtain the *receiver operating characteristic* (ROC), a curve that measures the classifier performance independent on the threshold value. The *area under curve* (AUC) [14] can be used as a decision class prediction statistic. The closer the AUC value to 1, the better the decision class is predicted. Value of 0.5 means that the classifier is performing no better than chance.

3 METHODS

3.1 Data

Let us provide a brief introduction to protein structure. Proteins are chains of amino acids (also called residues). There is a number of known amino acids; in higher organisms, 20 amino acids, each of unique structure, constitute basic blocks building proteins. The amino acids in proteins, apart from proline, consist of a carboxylic acid ($-COOH$) and an amino ($-NH_2$) functional group attached to the same tetrahedral carbon atom. This carbon is called the α-carbon (later referred to as C_α). Distinct R-groups, that distinguish one amino acid from another are attached to C_α. An exception is the glycine residue where the R-group is a hydrogen. The fourth substitution on the tetrahedral α-carbon of amino acids is hydrogen (Figure 3.1). A protein may consist of a few tens to approximately a thousand amino acids, therefore there is a great diversity of possible protein sequences. The linear chains fold into specific three-dimensional conformations, which are determined by the sequence of amino acids.

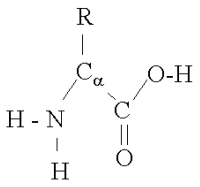

Figure 3.1. Amino acid structure

A structural protein domain is an element of overall structure that is self-stabilizing and often folds independently of the rest of the amino acid chain. Many domains are not unique to one protein and can appear as parts of various proteins. Domains are often associated and named with a protein function since these are the parts in protein responsible for its function.

All the proteins of solved structure are kept in the Protein Data Bank (PDB) database. The *PDB file format* is a special format for representing protein structures as a sequence of atoms and their coordinates. The Structural Classification of Proteins (SCOP) [25] database is a classification of PDB database proteins based both on their sequential and structural similarities, along with functional and mechanistic information. The classification organizes the domains in a special hierarchy according to their evolutionary relationship.

3.1.1 Input data

All the analysis was performed on a subset of ASTRAL [8], version 1.67, protein domains with less than 40% sequence identity to each other. The ASTRAL compendium provides selected subsets of representative protein domains, without redundancy. The domains are derived based on the SCOP database classification. The whole ASTRAL set contained 6600 domains. Similarly to Kryshtafowych and Fidelis [23], only the descriptors of at least three

segments were taken into account. There were $1,040,860$ such descriptors found for the whole set. Due to the time limitations, the system was tested and analysis was performed only on a subset of 657 protein domains and $148,012$ descriptors found in those domains. To assure the proper distributions of Gene Ontology (see Section 3.1.2) annotations in the subset, first there were 27 annotations chosen and subsequently the domains annotated by those were taken.

3.1.2 Protein functions

The Gene Ontology (GO) is a project that aims at providing consistent descriptions of genes and genes products. There are three distinct ontologies (structured vocabularies) developed:

- **molecular function** - describes activities, such as catalytic or binding activities, at the molecular level;
- **biological process** - accomplished by one or more ordered assemblies of molecular functions;
- **cellular component** - refers to the part of the cell, i.e.: compartment, anatomical structure or a gene product group, associated with the activity of the gene product in question.

Each of the ontologies forms a directed acyclic graph (DAG), i.e. each vertex may have several ancestors. Annotation of a gene product with a descendant attribute implies that it holds all attributes of its ancestor.

The GO classes from the molecular function ontology were used to annotate the set of domains. Each descriptor is associated with functions of its protein of origin. While annotating the domains with their GO classes, the DAG form of the ontology had to be taken into account in order to choose the functions from the proper graph level. The functions taken should not be too specific nor too general. The algorithm and software proposed by Hvidsten et al. [16] was applied.

3.2 Local descriptor method

The method for assembling and comparing descriptors from proteins was introduced by Kryshtafowych and Fidelis [23]. In the following sections the definition of local descriptor and some other corresponding concepts are covered. Subsequently, the algorithm for comparing and grouping descriptors is briefly presented. In this thesis a mathematical formalization of all the concepts is introduced.

3.2.1 The concept of a local descriptor

As mentioned before, local descriptor is a small protein substructure that consists of short continuous backbone segments that are close in 3-dimensional space but not necessarily

along the protein sequence. The segments are centered around one protein residue and they consist on average of 5-7 residues. When naming a descriptor, it is sufficient to provide the protein/domain name and the residue number. An example of a descriptor centered around the 83^{rd} residue in protein 1A33 is presented in Figure 3.2.

Figure 3.2. Protein 1A33 with a local descriptor centered on residue 83 (1A33_#83), the centre residue is distinguished.

The algorithm for calculating descriptors is as follows:

For a given residue r in a given protein a set of close residues is computed. Each residue is represented as a point on a vector $[C_\alpha, C_\beta]$ at the distance of 2.5 from C_α. In case of glycine, which does not possess a C_β atom, the C_α atom coordinates are taken. A pair of residues is close when the Euclidean distance between them is less than 6.5. The choice of the distance parameters was taken from [23].

Each of the close residues is added to the descriptor together with its four sequence neighbors, two from each side. This five-residues-long chain is called an *element*. The overlapping element sequences are joined to form *segments*.

Definition 3.2.1. Formally a *descriptor* d is a tuple $\langle r_c, R, E, S\rangle$ where:

- r_c is a central residue
- Let N be a set of close structural neighbors of r_c, $r_c \in N$. R is a set of the closest sequential neighbors (i.e. two residues from each side on the sequnce) of the residues from the set R and the set N itself, i.e $N \subseteq R$.
- elements and segments of the descriptor are the ordered sets of residues from the set R; E is the set of elements and S is the set of segments.

For a given structure a descriptor for each residue is computed.

3.2.2 Grouping descriptors

Let D be a set of descriptors calculated for the given set of proteins. The next step is to extract the most common geometrical conformations. In order to accomplish this task,

all pairs of descriptors are compared and for each descriptor a set of similar descriptors is assembled. Comparing pairs of descriptors from the whole database results in a set of groups, one group for each descriptor. The process of assembling groups consists of two stages: comparison of the descriptors which leads to the formation of the *preliminary groups* and the so called *cleaning of the groups*. The most common conformations are those of descriptors with the biggest groups.

Comparing descriptors

Root means square distance ($RMSD$) is a measure that is commonly used for evaluation of protein structures similarity. The $RMSD$ score is calculated from a sequence of coordinates $\{a_i\}$ and $\{b_i\}$ for the two given molecules, both of length n:

$$RMSD = \sqrt{\frac{\sum(a_i - b_i)^2}{n}} \tag{3.1}$$

Before calculating the $RMSD$ score, the molecules need to be superimposed in an optimal way, in order for the resulting score to be minimized. Superimposing affects the coordinates only in terms of rotation operations which does not affect the relative distances between the atoms in the molecule. There were several superimposing algorithms proposed, like Kabsch [19], McLachlan [24] or Kearsley [21].

Descriptors are compared using a similarity function

$$S : \mathrm{D} \times \mathrm{D} \rightarrow \{true, false\} \tag{3.2}$$

The function is described in detail in [23]. This thesis discusses only some of its key properties. The value of the function is computed based on the following descriptor properties:

- number of elements and segments,
- shapes of segments,
- $RMSD$ scores between the descriptor elements and segments,
- overall $RMSD$ score between the descriptors.

While comparing a pair of descriptors, the best alignment is also searched for. The elements from the first descriptor are mapped to the elements of the second descriptor. By obtaining the elements mapping, it can be derived which residues correspond to each other and the alignment can be constructed. The alignment might incorporate some gaps, this is due to the fact that not all of the elements from the first descriptor had to be mapped and the descriptor sizes might differ. The similarity function is not symmetric, i.e. given two descriptors d_i, d_j such that descriptor d_j belongs to the group of descriptor d_i does not imply that descriptor d_i belongs to the group of d_j, and vice versa. This can be formally denoted as:

$$S(d_i, d_j) \nLeftrightarrow S(d_j, d_i),$$

Moreover, even if $S(d_i, d_j) = S(d_j, d_i) = true$, the resulting alignments are not necessarily identical.

Preliminary groups and cleaning of the groups

Definition 3.2.2. Formally a *preliminary group* PG_d of descriptor d is a tuple $\langle d_{seed}, P_d, M_\delta \rangle$ where:

- $d_{seed} = \langle r_{seed}, R_{seed}, E_{seed}, S_{seed} \rangle$ is the seed descriptor,
- $P_d \subseteq D$ is a set of descriptors *similar* to the seed descriptor d_{seed}
- M_δ is a set of injective, partial functions δ defined for each descriptor $d_i \in P_d$. Take descriptor $d_i = \langle r_{ci}, R_i, E_i, S_i \rangle$. Function δ_i represents a mapping of elements from E_i to the elements of the seed descriptor d_{seed}, i.e $\delta_i : E_i \to E_{seed}$.

The last step, the so called "cleaning" of the groups, involves rearrangements of the descriptors within the groups. Take a preliminary group $PG_d = \langle d_{seed}, P_d, M_\delta \rangle$. Having the elements of d_{seed} mapped in the previous step, one can observe that some parts of the seed descriptor are rarely mapped and are uncommon for the other group members. This indicates that they are probably too specific and should not belong to the representative descriptor structure. These parts are therefore removed from the seed descriptor.

Subsequently, the member descriptors belonging to P_d are processed. All elements that were not mapped or just lost their mappings are also being removed. Once processed, some of the descriptor segments could have been shortened or even removed. All the descriptors that have number of segments different than the cleaned seed descriptor, are sieved out from the group.

Definition 3.2.3. A *cleaned descriptor* c is a tuple $\langle d, R_{cleaned}, E_{cleaned}, S_{cleaned} \rangle$ where:

- $d = \langle r_c, R, E, S \rangle$ is the original descriptor
- $R_{cleaned}$ is the set of residues of the cleaned descriptor, $R_{cleaned} \subseteq R$
- $E_{cleaned}$ is the set of the remaining elements, $E_{cleaned} \subseteq E$
- $S_{cleaned}$ is the set of the cleaned descriptor segments obtained by joining the overlapping elements from $E_{cleaned}$.

Let C be a set of "cleaned" descriptors. Let us introduce an auxiliary notion of the *cleaning transformation*

$$Clean : \mathrm{D} \times \mathrm{D} \to \mathrm{C} \cup \{nil\}, \tag{3.3}$$

where $Clean(seed, d)$ is a cleaned descriptor from the set C. If the descriptor d does not belong to the cleaned PG_{seed} then nil is returned.

Since cleaning does not change alignments in any other way than by removing insignificant, rarely present parts, the alignments of the remaining residues can be used for computing the $RMSD$ score. Similarly as in [23], only the C_α coordinates are taken into account when computing the score. The $RMSD$ score is normalized by the number n of corresponding residues, the resulting measure $RMSD_{adj}$ for the cleaned descriptor c and an arbitrary cleaned c_i belonging to the cleaned group PG_d is calculated as follows:

$$RMSD_{adj}(c, c_i) = 3 * \frac{RMSD(c, c_i)}{\ln n} \tag{3.4}$$

Therefore a preliminary group processed using the *Clean* transformation can be formally defined as:

Definition 3.2.4. A *cleaned group* G_d for the descriptor d is a tuple $\langle d, c, P_c, M_{RMSD} \rangle$ where:

- $d \in \mathrm{D}$ is the original descriptor
- $c = \langle d, R_{cleaned}, E_{cleaned}, S_{cleaned} \rangle$ is the cleaned seed descriptor
- P_c is the set of the cleaned member descriptors, note that for all $c_i \in P_c$ where $c_i = \langle d_i, R_i, E_i, S_i \rangle$, $|S_i| = |S_{cleaned}|$
- M_{RMSD} is the set of $RMSD_{adj}$ scores for all $c_i \in P_c$, $M_{RMSD} : \mathrm{C} \to \mathbb{R}$

Both the cleaned seed descriptor and the cleaned member descriptors from G_d are the substructures of the original descriptors and can be referred to only with association to the group they belong to. It is worth mentioning, that the cleaned group members final shape depends on other group members. Note that the cleaning transformation involves removing unpopular elements from the group members, where the "unpopularity" is defined by some constants. Adding a new descriptor might change the situation. Therefore, if we want to add a new group member, we have to add it to the preliminary group and then repeat the cleaning transformation.

3.2.3 Dissimilarity function

On a basis of $RMSD_{adj}$ a function $Diss : D \times D \to \mathbb{R} \cup \{\infty\}$ measuring the descriptors dissimilarity can be defined:

$$Diss(d_i, d_j) = \begin{cases} RMSD_{adj}(c_i, c_j) & \text{if } c_j \in P_c; \\ \infty & \text{otherwise.} \end{cases} \tag{3.5}$$

where $G_{d_i} = \langle d_i, c_i, P_c, M_{RMSD} \rangle$ is the group for descriptor d_i, $c_i = Clean(d_i, d_i)$ and $c_j = Clean(d_i, d_j)$.

A *metric* on a set X is a function (called the distance function or simply distance) $d : X \times X \to \mathbb{R}$ (where $\mathbb{R}$ is the set of real numbers). For all x, y, z in X, this function is required to satisfy the following conditions:

1. $d(x, y) \geq 0$ (non-negativity)
2. $d(x, y) = 0 \Leftrightarrow x = y$ (identity of indiscernibleness)
3. $d(x, y) = d(y, x)$ (symmetry)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

Diss is not a metric in a proper sense, the following conditions may not hold:

- *symmetry* $Diss(d_i, d_j)$ does not always equals $Diss(d_j, d_i)$. For instance, if the cleaned d_j does not belong to G_{d_i} and the cleaned d_i belongs to G_{d_j}, than
  $Diss(d_i, d_j) = \infty$ and $Diss(d_j, d_i) < \infty$.
- *triangle inequality.* As a counterexample consider the following situation:
  Take $d_x, d_y, d_z \in D$ such that the cleaned d_z does not belong to G_{d_x}, whereas cleaned $d_y \in G_{d_x}$ and cleaned $d_z \in G_{d_y}$.
  Then $Diss(d_x, d_z) = \infty$ and $Diss(d_x, d_y) + Diss(d_y, d_z) \leq \infty$
  therefore
  $Diss(d_x, d_z) \not\leq Diss(d_x, d_y) + Diss(d_y, d_z)$

The *non-negativity* and the *identity of indiscernibleness* conditions do hold.

3.2.4 The local descriptor method summary

The algorithm gets a set of proteins as input, which are processed in order to extract the local descriptors. Once the descriptors are found, they are compared to each other, and for each of them the so called *preliminary group* (Section 3.2.4 and Definition 3.2.2) of similar descriptors is gathered. Subsequently, the groups are *cleaned* (Section and Definition 3.2.4) and the within group $RMSD$ distances are computed. The result of the whole algorithm is an asymmetric dissimilarity function $Diss$ (Section 3.2.3) returning a dissimilarity score for an arbitrary pair of descriptors.

3.3 Clustering the local descriptor data

The goal of the clustering stage is to provide an information system for the prediction stage, when the rule based classifier is constructed. Each cluster corresponds to an attribute in the decision table, the decision class values are the GO classes. The clustering is supposed to result in a set of clusters, where cluster is a set of descriptors, represented by one of the member descriptors.

Definition 3.3.1. Formally, a *clustering* of the set of descriptors D is a partition of D into n sets $\langle (C_1, rep_1), \ldots, (C_n, rep_n) \rangle$ such that:

1. $\bigcup_{i=1}^{n} C_1 = D$
2. for all $1 \leq i, j \leq n, C_i \cap C_j = \varnothing$
3. for each $1 \leq i \leq n, rep_i \in C_i$ is a representative member of the cluster.

Notice that the groups obtained from the previous step, apart from the disjunction criterion, could also serve as clusters. Each group would be represented by its seed descriptor, which together with its group members would constitute a cluster. The disjunction criterion is not satisfied since the groups may overlap, however this criterion is not really crucial in this application. The major drawback that makes the clustering step necessary is the very big number of groups. It is a basic principle in data mining that the bigger the

number of attributes, the harder the learning problem is. The so called ill-defined problems are unsuitable for the majority of the supervised learning algorithms. This phenomenon is often referred to as the *curse of dimensionality* [3]. By clustering, some redundant information (for example joining groups that mostly overlap in one cluster) and noise in data (very small groups of uncommon descriptors) can be removed.

When choosing the algorithms, big size of the dataset had to be taken into account. The algorithms of high complexity could not be applied. On the other hand the dissimilarity matrix appeared to be very *sparse*, i.e. the majority of records is ∞. Descriptors $d_1 \in D$ and $d_2 \in D$ are "comparable" only if $d_1 \in G_{d_2}$ or $d_2 \in G_{d_1}$. Otherwise dissimilarity of these two is equal to ∞. However, the data is still too large to fit in to the main memory and the whole clustering had to be implemented as a series of I/O, database operations.

Two clustering algorithms were applied, an agglomerative, single linkage hierarchical algorithm and a density based, DBSCAN algorithm. Both algorithms require little prior knowledge about the dataset to be clustered: one does not specify the number of clusters in advance. This was a major drawback of the partitioning-like algorithms, due to the large size of the dataset the number of clusters is very hard to estimate. Moreover, the computation complexity of the K-medoids algorithm was prohibitive, especially that the algorithm would have to be run numerous times in order to assess the appropriate number of clusters. The K-means algorithm, of lower complexity, was not suitable for the dissimilarity matrix data representation.

3.3.1 Hierarchical algorithm

The single linkage version of the hierarchical algorithm was implemented. The reason for this choice was the dissimilarity matrix $Diss$ sparsity. Consider the following situation. Let C_1 and C_2 be clusters and assume that there exist descriptors $d_1 \in C_1$ and $d_2 \in C_2$ such that $Diss(d_1, d_2) = \infty$. Then, according to both the average and the complete linkage version $DistClust(C_1, C_2) = \infty$. The probability of such situation is very high since only 0.036% of the descriptor pairs have a dissimilarity defined to be less than ∞. The computational cost of the single linkage version is lower than the one of the average and complete linkage versions. It is only required that a single dissimilarity between two objects $c_1 \in C_1$ and $c_2 \in C_2$ was small for two clusters C_1 and C_2 and hence it does not have to go through the other cluster members.

Algorithm 1 presents the framework of the hierarchical descriptor clustering. The algorithm stops when there are no cluster distances less than ∞. The dendrogram obtained is a forest instead of a single tree because not all the pairwise dissimilarities are defined. When merging the clusters, the new cluster representative descriptor is chosen among the representative descriptors of the clusters being merged; the representative descriptor having the bigger group of similar descriptors is taken. The new cluster is positioned one level higher (closer to the root) than the cluster in the previous iteration step.

Termination

The stop condition is satisfied: the while loop stops because at each iteration the size of the set of clusters decreases: one cluster is added (created in step 11 and added in step 19)

Algorithm 1 $HierarchicalClustering(D, Diss)$

Require: D set of descriptors, unclassified to any clusters, dissimilarity matrix on descriptors $Diss$
Ensure: A dendrogram of points representing clusters

1: $clusters \leftarrow \varnothing$
2: **for all** descriptors $d_i \in D$ **do**
3: $C_i.representative = d_i$;
4: $C_i.level \leftarrow 0$;
5: add C_i to $clusters$;
6: **end for**
//DissClust is a matrix representing clusters dissimilarities
7: $DissClust \leftarrow Diss$
8: $level = 1$;
9: **while** $\exists C_1, C_2 \in clusters$ such that $DissClust(C_1, C_2)) < \infty$ **do**
10: select clusters C_i and C_j such that $DissClust(C_i, C_j)$ is minimal
11: merge clusters C_i and C_j into one cluster C
//set level C
12: $C.level \leftarrow level$;
13: $level \leftarrow level + 1$
14: **if** $C_i.groupSize > C_j.groupSize$ **then**
15: $C.representative = d_i$
16: **else**
17: $C.representative = d_j$
18: **end if**
19: add column and row to matrix $DissClus$ for cluster C;
20: $UpdateDissimilarities(DissClust, C, C_i, C_j)$;
21: remove from $DissClust$ columns and rows corresponding to clusters C_i and C_j.
22: **end while**

Algorithm 2 $UpdateDissimilarities(DissClust, C, C_i, C_j)$

Require: $DissClust$ dissimilarity matrix for clusters, cluster C resulting from merging clusters C_i and C_j
Ensure: updated matrix $DissClust$

1: **for all** clusters C_k **do**
2: $DissClust(C_k, C) = \min(DissClust(C_k, C_i), DissClust(C_k, C_j))$;
3: $DissClust(C, C_k) = DissClust(C_k, C)$
4: **end for**

to the set but two are removed (step 19).

Complexity

The pairwise dissimilarities are kept in a database table that has an index on the distance column (see Chapter 4 and Figure B.4), therefore extracting the closest descriptors can be done in constant time, $O(1)$. Let C_i and C_j be the merged clusters. The distances table can be updated in $O(n)$ time. For each cluster the distances to C_i, C_j are replaced with

the distance to the cluster to be merged. There are up to $n-1$ merging steps, hence the complexity of the algorithm is $O(n^2)$.

3.3.2 DBSCAN algorithm

DBSCAN algorithm (*A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*) [10] was designed to deal with spatial data i.e. data related to space, such as satellite images and protein structure data [10, 26]. The idea of the algorithm is that for each clustered point an area within the specified radius ε should contain at least a minimum number of points, $MinPts$. In other words, the density of the neighborhood of the point should be bigger than a certain threshold.

The algorithm is based on the notion of two points being *density-reachable* with regards to parameters ε and $MinPts$. The ε-neighborhood of point p , $N_\varepsilon(p)$ is the area of radius ε and centre in point p. Take two points, p and q. Point q is *directly reachable* from p if q belongs to the ε-neighborhood (i.e. to the area of ε radius) of p. In turn the points are *density-reachable* if there exists a path p_1,..., p_k, where $p = p_1$ and $q = p_q$ such that p_{i+1} is directly reachable from p_i.

Application to the local descriptor data is straightforward; the descriptor neighborhood is included in its group of similar descriptors, and the distances are provided by the $Diss$ function. Algorithm 3 presents the DBSCAN algorithm adjusted to the descriptor data. The algorithm stops when all the descriptors are assigned to some cluster, some of them belong to the cluster *noise* of structurally uncommon descriptors. Algorithm 4 presents the $ExpandCluster$ function. Assume that the cleaned group of d is $G = \langle d, c, P_c, M_{RMSD} \rangle$. The neighborhood of radius ε of descriptor d is a set $N_\varepsilon = \{d_i : c_i = \langle d_i, R_i, E_i, S_i \rangle \in P_c \ \wedge Diss(d, d_i) < \varepsilon\}$. The descriptor is said to be in the *core* [10] of its cluster when its neighborhood size $|N_\varepsilon|$ is bigger the the parameter $MinPts$. Otherwise the descriptor is situated on the boundary of the cluster. As a cluster representative descriptor chosen is the *core* descriptor that has the biggest group of similar descriptors among all the cluster members.

Algorithm 3 $DBSCAN(\mathrm{D}, Diss, \varepsilon, MinPts)$

Require: D - set of descriptors unclassified to any clusters, dissimilarity matrix **Diss**, DBSCAN parameters Eps and $MinPts$
Ensure: Points assigned to clusters, noise cluster

```
clusterID = 1;
for all descriptors d_i ∈ D do
    d_i.IsCorePoint ← false;
end for
while exists unclassified point d_i do
    ExpandCluster(d_i, clusterID)
    clusterID ← clusterID + 1;
end while
```

Algorithm 4 $ExpandCluster(d_i, clusterID)$

```
1: seeds ← GetCleanedGroupMembers(d_i, Eps);
   //get all the d_i group members that are closer to d_i than the Eps parameter
2: if seeds.size() < MinPts then
3:    AssignCluster(d_i, NOISE);
4:    return false;
5: end if
6: AssignCluster(seeds, clusterID);
7: AssignCluster(d_i, clusterID);
8: d_i.IsCorePoint ← true;
9: while seeds not empty do
10:   currentD ← seeds.front()
11:   result ← GetCleanedGroupMembers(currentD, Eps);
12:   if result.size() ≥ MinPts then
13:      currentD.IsCorePoint ← true;
14:      for all descriptors d_k in result do
15:         clID ← GetGroupCluster(d_k);
16:         if clID = UNCLASSIFIED or NOISE then
17:            if clID = UNCLASSIFIED then
18:               seeds.append(d_k)
19:            end if
20:            AssignCluster(d_k, clusterID);
21:         end if
22:      end for
23:   end if
24: end while
25: seeds.delete()
26: return true;
```

Termination

The main loop iterates through the set of unclassified descriptors, while processing the descriptor it is always being assigned to some clusters (step 3 or 7 in the *ExpandCluster* function, Algorithm 4). Therefore the stop condition is satisfied.

Complexity

In the main loop only the unclassified descriptors are processed by invoking the *ExpandCluster* function. The descriptor and all the descriptors density-reachable from it are being assigned to the clusters (therefore they will not be processed in the main loop). For each of the descriptors density-reachable from the processed descriptor the *GetCleanedGroupMembers* function is invoked. The function returns the descriptor neighborhood N_ε, which can be done in logarithmic time, $O(log(n))$. Since the relation of being density-reachable is asymmetric it might happen that for some point the *GetCleanedGroupMembers* function will be called more than once. However the algorithm authors [10] claim that the ε-neighborhoods are expected to be much smaller then the whole dataset being clustered. Hence, even though some descriptors might be processed several times, this fact is insignif-

icant when approximating the algorithm complexity, which is $O(nlog(n))$.

3.3.3 Asymmetric dissimilarity matrix

An interesting aspect was the asymmetric dissimilarity matrix (see Section 3.2.3). In case of the DBSCAN algorithm this fact introduced indeterminacy, the clusters assignment could rely on the order of processing. Consider the situation depicted in Figure 3.3. Here the descriptors are represented by the small circles, the arrows indicate that the pointed descriptor belongs to the ε-neighborhood of the pointing descriptor. Depending on the starting point, a or b, we might end up with one or two clusters. If a is processed first, cluster b is assigned to the cluster of a and it is not touched any more. If the algorithm starts with b it adds all the neighbors of b to its cluster and then it proceeds to unclassified a. Descriptor b is already classified therefore it will not be added to the cluster of a.

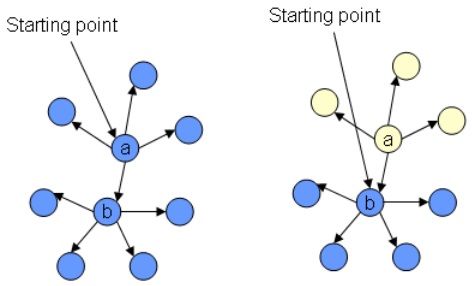

Figure 3.3. DBSCAN clusters indeterminacy caused by the dissimilarity matrix asymmetry.

In the hierarchical algorithm case the asymmetry was also problematic. The pair-wise cluster dissimilarities are computed on a basis of the descriptors dissimilarities, hence it had to be stated how to deal with the asymmetry. The default algorithm performance was to ignore it and take a pair of clusters that is closest in at least one direction.

Hastie et al. [15] suggest that since most clustering algorithms require the dissimilarity matrix to be symmetric, the solution is to replace the original matrix $Diss$ with the matrix $\frac{Diss+Diss^T}{2}$. Since in this case most matrix entries were ∞, then the recommended transformation could lead to loss of information: imagine that $Diss(d_1, d_2) = \infty$ and $Diss(d_2, d_1) = d$ where $d < \infty$. The entry for dissimilarity between d_1 and d_2 in matrix $\frac{Diss+Diss^T}{2}$ would be $\frac{d+\infty}{2} = \infty$. Therefore the case in which both dissimilarities are infinite would not be distinguished from the described case. To prevent the loss of information, the following steps were applied:

1. all the ∞ entries in $Diss$ were replaced with constant MAX, that was bigger than any other finite entry,
2. matrix $Diss$ was replaced with $\frac{Diss+Diss^T}{2}$,
3. the entries that were bigger than MAX were again replaced with ∞.

The algorithms were run for both the original $Diss$ matrix and the symmetric one. See Chapter 5 for the performance comparison.

3.4 Evaluating clustering quality

Comparing, grouping and clustering descriptors takes into account only their structural similarity. The information about the functions of proteins is disregarded during the whole process. The assumption was that the clusters should inherently group descriptors belonging to the proteins of similar functions.

Gibbons and Roth [13] dealt with a similar problem when clustering a microarray data. They proposed a method, based on the mutual information concept, for measuring the correlation between the cluster membership and the Gene Ontology annotations of the clustered genes. However their method can be applied to any type of data that can be represented as a decision table $\mathbb{A} = (U, \{d\})$, with only the decision attribute. The objects in U are clustered on the grounds of some other features, the decision attribute is not taken into account. The method can help in stating whether there exist a correlation between the cluster membership and the decision classes or in adjusting the clustering algorithm parameters.

3.4.1 Mutual information and entropy

The mutual information is a measure for assessing the dependence of random variables. First the concept of *entropy*, suggested by Claude E. Shannon [31] is provided.

Suppose we have a random variable X over the discrete set of events whose probabilities of occurrence are $p_1, p_2, \ldots, p_n$ where $\forall i, p_i > 0$ and $\sum_{i=1}^{n} p_i = 1$. We want to measure the amount of information carried by the occurrence of a single random event. The alternative way to look at it is to treat it as a measure for the rate of randomness in a random event. The intuition while defining the entropy was as follows: *The less probable the occurrence of the event is, the more information it carries* [32].

Definition 3.4.1. Let X be a random variable and p be a probability density function, i.e $p(x) > 0$ and $\sum_{x \in X} p(x) = 1$. The *entropy* of X is given by the formula:

$$H(X) = \sum_{x \in X} p(x) log(\frac{1}{p(x)}) = -\sum_{x \in X} p(x) log(p(x))$$

H takes on its maximum value when all the events probabilities are equal, i.e. $\forall x_i, x_j \in X : p(x_i) = p(x_j)$.

The mutual information of two random variables is a quantifies the independence of the variables. Informally, mutual information measures the information of X that is shared by Y. If X and Y are independent, then X contains no information about Y and vice versa, so their mutual information is zero. If X and Y are identical then all information conveyed by X is shared with Y. Any knowledge about X reveals nothing about Y therefore the mutual information is equal to the entropy of X.

Definition 3.4.2. Let X and Y be discrete random variables. Let p with $p(x, y) = Pr(X = x, Y = y)$ be the joint probability density function of X and Y, f with $f(x) = Pr(X = x)$ be the probability density function of X alone, and g with $g(y) = Pr(Y = y)$ be the probability density function of Y alone. The *mutual information* of X and Y is

given by $I(X,Y)$, defined as follows for the discrete case:

$$I(X,Y) = \sum_{x,y} p(x,y) \times \log \frac{p(x,y)}{f(x)\, g(y)}.$$

If X and Y are independent then $I(X,Y) = 0$, since $p(x,y) = f(x)g(y)$ and hence $\log \frac{p(x,y)}{f(x)\, g(y)} = \log 1 = 0$

The mutual information can be equivalently expressed as:

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y) \qquad (3.6)$$

The mutual information is therefore symmetric (i.e. $I(X,Y) = I(Y,X)$). As $H(X) > H(X|Y)$, and nonnegative (i.e. $I(X,Y) \geq 0$).

Assume that $Y_i, 1 \leq i \leq n$ are random variables. One can define a multidimensional mutual information between a random variable X and the set of n random variables Y_i as $I(X, Y_1, ..., Y_n)$. If all the random variables Y_i are pairwise independent, then [13]:

$$I(X, Y_1, ..., Y_n) = \sum_{i=1}^{n} I(X, Y_i) = nH(X) + \sum_{i=1}^{n} H(Y_i) - \sum_{i=1}^{n} H(X, Y_i) \qquad (3.7)$$

3.4.2 Method description

Assume C is a random variable representing the clustering of k clusters: the i^{th} event is the i^{th} cluster membership, the probability of the event can be computed as a fraction of the i^{th} cluster size to the whole space size. Similarly assume that D_i is a random variable representing the i^{th} decision class. Here the probability space has two events: belonging to the i^{th} decision class and not belonging to the i^{th} decision class. The probability p_{d_i} of the decision class membership is again a fraction of the decision class size to the whole space size. The probability of the contrary event is $1 - p_{d_i}$.

Contingency table

A contingency table [29, 9] is a table showing the responses of subjects to one variable as a function of another variable, i.e. rows are labeled by the values of one variable and columns by the values of the other one. The entries, nonnegative integers, give the size of the sets intersection, i.e. number of objects that have a certain combination of features. For example the contingency table 3.1 shows possession of a protein function (decision class membership) as a function of cluster membership. The probability of each of the events can be computed from the table as a fraction of the corresponding table entry and the whole space size.

Table 3.1. Contingency table

	d_i	$\neg d_i$	*total*
c_1	6	5	11
c_2	2	8	10
c_3	1	10	11
c_4	10	4	14
total	19	27	46

Mutual information

Descriptors are associated with functions of their host proteins, there might be more than one function associated with a descriptor. Having this information the contingency tables are constructed, one for each possible function (Table 3.1). The aim is to assess a mutual information, $I(C, D_1, \ldots, D_k)$ value reflecting the association of the clustering C and all the functions $D_1, ..., D_k$. In here we assume that the variables $D_1, ..., D_k$ are pair-wisely independent. Hence, the mutual information can be computed as in Equation 3.7:

$$I(C, D_1, \ldots, D_k) = \sum_{i=1}^{k} I(C, D_i) = kH(C) + \sum_{i=1}^{k} H(D_i) - \sum_{i=1}^{n} H(C, D_i) \tag{3.8}$$

Assume that clustering C results in n clusters. The contingency tables provide the probability distribution, hence the entropies $H(C, D_i)$ can be computed. Let p_j be the probability of the cluster j members possessing the i^{th} function, $q_j = 1 - p_j$ is the probability of the contrary event. Then the entropy $H(C, D_i)$ is computed as follows:

$$H(C, D_i) = -\sum_{j=1}^{n} p_j log(p_j) + q_j log(q_j). \tag{3.9}$$

Let p_{c_i} be the probability of the i^{th} cluster membership, $\sum_{i=1}^{n} p_{c_i} = 1$. Then the $H(C)$ entropy is computed as

$$H(C) = -\sum_{i=1}^{n} p_{c_i} log(p_{c_i}). \tag{3.10}$$

Finally, let r_i be the probability of possessing the i^{th} function. The formula for entropy $H(D_i)$ is:

$$H(D_i) = -(r_i log(r_i) + (1 - r_i) log(1 - r_i)). \tag{3.11}$$

The higher the value of mutual information the more significant is the correlation. This measure can be used when comparing performance of clustering algorithms or to assess how the obtained cluster membership differs from the random cluster assignment. In order to do that, one has to perform a set of random cluster assignments and compute the mutual information value for them. Having the distribution of the randomly obtained mutual information values, the mean and standard deviation, I_{random} and δ_{random}, can be computed. A z-score [13] measures the distance of the clustering from the random one. The formula is:

$$z = \frac{I_{real} - I_{random}}{\delta_{random}}. \tag{3.12}$$

The higher the z-score, the further from random is the clustering. The mutual information is nonnegative therefore the z-score is positive if $I_{real} > I_{random}$. Note that if the clustering obtains a negative z-score this means that its performance is worse than random.

3.5 Classifier

Based on the clustering of the set of descriptors, the predictive model is constructed. Take a clustering $C = \langle (C_1, rep_1), \ldots, (C_n, rep_n) \rangle)$(see Definition 3.3.1). Let Rep be a set of the representative descriptors $\{rep_1, \ldots, rep_n\}$. The decision table used for training the classifier is $\mathbb{A} = (U, Rep \cup \{function\})$, where:

- U is a set of proteins,
- $\forall rep_i \in Rep, V_{rep_i} = \{0, 1\}$,
- $V_{function}$ is a set of gene ontology annotations.

All the descriptors D where extracted from the proteins from U. The values of the features are either 1 or 0, depending on whether the protein possess a descriptor that is present in the cluster corresponding to the attribute. The decision attribute corresponds to the function of the protein. There might be more than one GO annotation for certain protein, in such situation there is a row added to the decision table for each of the annotations.

3.5.1 Predicting protein function

The classifier can be applied to the unseen (not present in the training set) proteins. To this end a protein has to be represented in a proper way, as a vector of features. Each of the features corresponds to one of the clusters of common local descriptor conformations. Take the decision table $\mathbb{A} = (U, Rep \cup \{function\})$ used as a training set during the classifier construction. To obtain the vector, Algorithm 5 is applied.

Set D_I is a set of descriptors found in the investigated protein. In order to determine whether the protein incorporates the common 3-dimensional conformations, the descriptors from D_I are compared to the cluster representative descriptors. Instead of the original descriptors, the cleaned versions of the representative descriptors are used. The cleaned descriptors have the most uncommon parts removed and are therefore less specific. If for a certain representative descriptor a similar descriptor is found, then the protein gets value 1 for that feature, otherwise it gets 0.

The *Compare* function for stating whether two descriptors are similar is based on the similarity function (3.2). First, the seed elements are compared and if the $RMSD$ score of those is small enough, other elements are being compared in order to find the best matching and hence alignment. If the $RMSD$ score for the whole molecules is less than a certain threshold, the descriptors are assessed to be similar. As a threshold value the average cluster diameter is taken.

Subsequently, a classifier $\hat{d}$ (Definition 2.2.14) is applied to the protein and the prediction is returned.

Algorithm 5 $ConstructFeatureVector(\mathbb{A}, p)$

Require: a decision table $\mathbb{A} = (U, Rep \cup \{function\})$, an investigated protein p (pdb format file)

Ensure: a vector vec of features from Rep for protein p

1: $D_I \leftarrow$ set of descriptors found in p
2: **for all** descriptors $rep_i \in Rep$ **do**
3: $G \leftarrow \langle rep_i, c_i, P_c, M_{RMSD} \rangle$ *//cleaned group of descriptor* rep_i
4: $vec[i] = 0$
5: **for all** $d \in D_I$ **do**
6: $b = Compare(c_i, d)$ *//compare the descriptors*
7: **if** b = true **then**
8: $vec[i] = 1$ *//descriptors were similar*
9: *break*
10: **end if**
11: **end for**
12: **end for**

4 IMPLEMENTATION

The system consists of two, quite independent modules. The off-line module is responsible for construction of the classifier, i.e. calculating, grouping and clustering descriptors and the decision table preparation. The classifier is applied in the on-line module, to predict functions of unseen proteins.

4.1 Off-line module

4.1.1 Data representation

The system was implemented in the C++ programming language as a library of classes. In general, the classes representing the data objects correspond to the definitions from Chapter 3.2. A diagram of classes used for data representation is presented in Figure B.1.

A protein structure is represented by the *Protein* class. The *Protein* class object stores a collection of atoms, keeping their sequence order. Atoms are represented by the *Atom* class that stores atom coordinates. The *Residue* class stores its C_α and C_β *Atom* objects, the residue centre coordinates (see Section 3.2.1), residue name, number, etc. Glycine residue is distinguished because it does not have C_β atom, and it is represented by the *GlyResidue* class deriving from the *Residue* class. For a given structure its descriptors are calculated by invoking the *Protein* class method, *calculateDescriptors*(). Descriptors are represented by the *Descriptor* class. A *Descriptor* class object contains a set of *Residue* class objects, which are organized into *Element* class objects. The *Element* class has a central *Residue* distinguished and an ordered vector of *Residue* objects. In this settings, the vector is always of length 5, however this is regulated by a parameter. By invoking a *Descriptor* class method, *JoinElements()*, a set of *Segment* class objects is calculated for a given *Descriptor* object. A *Descriptor* object has also its seed *Element* object distinguished.

The *DescriptorGroup* class represents both the preliminary and the cleaned group. The attributes that correspond to the preliminary group are: *seed* - a *Descriptor* class object and *descriptors* - a set of preliminary group members, each represented by the *DescriptorWrapper* class. The *DescriptorWrapper* class represents a descriptor in a group, it stores the descriptor, the group seed descriptor and the mappings of the elements of those two. By invoking a method *Clean()* the group cleaning is performed, where the *cleanedSeed* object and the cleaned group members are computed. First, new *DescriptorWrapper* class objects are constructed that have some elements removed, one for each descriptor member. Second, on a basis of the elements mapping an alignment is constructed for each *DescriptorWrapper* class object. Third, the *RMSD()* method of the *DescriptorWrapper* class is invoked to compute the cleaned descriptor and the cleaned seed dissimilarities. Finally, the cleaned descriptors are put to the *rmsdDistances* map, where each *Descriptor* class object is associated with its dissimilarity to the *cleanedSeed* object.

4.1.2 Command line programs

There are several independent applications responsible for each step of the process of the classifier preparation (see Figure 1.1):

- calculating descriptors from a given set of pdb format files,
- comparing two sets of descriptors against each other,
- loading a database with the comparison results,
- cleaning the groups, loaded in a database,
- two implementations of clustering the descriptors in a database,
- preparation of an information system.

Those applications employ the C++ library classes, they are implemented as command line programs.

Subsequently, the proteins from the information system are annotated with the Gene Ontology classes. The software used in [16] is used for that purpose. As a result a decision table is obtained. To construct a classifier, the Rosetta system, implementing the rough set based algorithm for inducing IF-THEN rules, is used.

4.1.3 Database

A MySQL relational database was used in the implementation. The database is used for storing:

- information about the descriptors,
- preliminary groups,
- cleaned groups,
- cluster assignments.

A diagram of tables used for representation of descriptors is presented in Figure B.2. A diagrams for representing the preliminary groups and the cleaned groups are shown in Figure B.3 and B.4 respectively.

The *Residue* table stores all the information concerning a residue, such as its C_α and C_β atoms coordinates, the residue centre coordinates, residue type and number in a structure. The *Element* table stores identifiers of the element seed residues. The *Residue_ Element_ M* table is responsible for mapping residues to elements. The *Descriptor* table stores a name, size, number of segments of a descriptor, identifiers of the seed element and of the seed residue. Elements are matched to descriptors by the *Descriptor_ Element_ M* table.

The *GroupP* table represents a preliminary group and stores an identifier of its seed descriptor. Descriptors are joined to their preliminary groups by the *GroupP_ Descriptor_ M* table. The *Element_ Element_ M* table stores element mappings for each preliminary group. One entry corresponds to one element *element* of the member descriptor *descriptor_ fk* that is mapped to some element *seedElement* from the seed descriptor of the preliminary group, *group_ fk*. The *Count_ Map* table counts how many times the element *element_ fk* from the seed descriptor of the preliminary group *group_ fk* was mapped by some elements of the group members.

The information about the cleaned descriptors has to be stored in a separate table *CDescriptor*. A cleaned descriptor is identified by its descriptor of origin (from the *Descriptor* table) and its cleaned group (from the *GroupD* table). Table *CDescriptor_ Element_ M* defines which elements constitute which cleaned descriptors. The *Group_ Descriptor_ M* table stores an entry for each *group_ fk* cleaned group member *descriptor_ fk*, together with the dissimilarity value *rmsd*.

A C++ API, MySQL++ was used in the C++ code to connect to the database and submit queries within the C++ programs. MySQL++ software is licensed under the GNU Lesser General Public License (LGPL).

4.1.4 Least squares fitting

The Bioinformatics Template Library [28] is a C++ library that implements numerous standard bioinformatics algorithms. The authors claim to provide an effective standard for the design of reusable software components for biocomputing. The library is licensed under LGPL.

The BTL library implements the Kearsley algorithm [21] for superimposing two 3-dimensional structures in order to find the best fit and compute the RMSD score. The algorithm was used in the descriptors grouping process.

4.1.5 Parallelizing the process

Comparing descriptors is very time consuming. Each comparison is supposed to result in the best elements superposition in terms of minimizing the $RMSD$ score. The algorithm implemented uses some heuristics [23] in order not to check all possible variants (average number of elements in a descriptor is $8,5$ which leads to $\approx 14,000$ variants, [23] computed from the Γ function), however it still involves checking around 100 superpositions which themselves involve solving constrained least-squares procedure of high time-complexity.

The comparison problem is yet very scalable. All pairs of descriptors need to be compared, but the comparisons are independent and do not affect each other. Hence the problem is very suitable for distributed systems. A computer grid, Swegrid (http://www.swegrid.se), was used to run the comparisons. Swegrid is a Swedish national computational resource, consisting of 600 computers in six clusters. It is designed mainly for *through-put computation*, that is to quickly process large numbers of loosely coupled non-parallel computations.

Preparing the grid tasks

The comparison problem is non-parallel, i.e. it can be divided into many subtasks that do not need to communicate with each other. Hence the most proper way of running the comparison on a grid was to run several independent tasks, and afterwards to join the separate results.

Assume that the dataset of descriptors is of size n. There are n^2 comparisons to perform. The straightforward approach would be to run n^2 tasks, one for each comparison. Doing so, the maximal parallelization is assured, nevertheless it is not optimal due to the amount of time spent on being assigned to one of the cluster nodes and the data transfer. The other solution could be to define n tasks, in which one of the descriptors is being compared to all the others. This is in turn not doable: due to the size of the descriptor dataset all the descriptors cannot be loaded to the main memory at the same time.

The solution adopted in here was to divide the set of descriptors into k parts of more or less equal size. Then for each pair of the subsets a task was defined.

Take the set D of all the descriptors and two arbitrary subsets, D_1 and D_2, where $D1 \subseteq D$ and $D_2 \subseteq D$. One task involves comparing all the descriptors from the set D_1 against all the descriptors from the set D_2. Note that as mentioned in Section 3.2.4, the similarity function for comparing two descriptors is asymmetric. This implies that comparing descriptors from the set D_1 against the descriptors from the set D_2 has to be done both ways. The program was implemented so that it both returns the groups for the set D_1 and D_2. Hence the number of tasks is the number of pairs in the k-elements set and the identities, $\binom{k}{2}+k$. The k parameter is chosen so that all the descriptors from the set $D_1 \cup D_2$ can be kept in main memory.

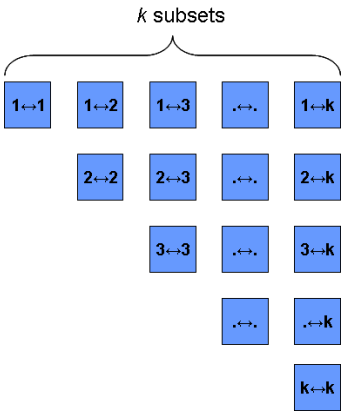

Figure 4.1. grid tasks

The most appropriate way of providing task input is through text files, which is faster than connecting to the database through network during the program execution. A special file format for describing descriptors and preliminary groups was developed. The whole process was as follows:

1. create files describing the descriptors, usually one file per protein domain
2. divide the files into k parts

3. define and submit $\binom{k}{2}+k$ tasks

4. load the output tasks files describing the groups

5. load the database with the tasks results.

Note that the parallelization concerns only the preliminary grouping. The problem of group cleaning is relatively less demanding. Whereas the preliminary grouping complexity is $O(n^2)$, the subsequent step requires $O(n \log(n))$ operations. Each of the preliminary groups has to be processed, gathering a preliminary group members can be done in logarithmic time. The problem of group cleaning is also well scalable, each preliminary group can be processed independently. However, contrary to the preliminary grouping case, here the data cannot be partitioned. Each of the preliminary groups can have members from the whole descriptor space, which cannot be kept in main memory at the same time. Hence the information concerning the preliminary groups has to be stored in a database which allows for loading the selected descriptors during the program execution. This makes the distributing of the group cleaning step more problematic, the reasonable solution would require grid resources with a common disk space.

4.2 On-line module

The online module has a client-server architecture with a web browser user interface. It was implemented using the Java struts technology and the Tomcat servlets container.

4.2.1 Struts

The Model-View-Controller design pattern is a software architectural approach for user-interactive applications. The application architecture is organized into three separate modules: the model with data representation, the view – user interface that enables the interaction and the controller module, which connects the remaining two modules. This kind of architecture facilitates interchange of user interface and model implementation.

Struts is a framework for the use of web-based server-side applications. It employs technology of Java beans, servlets and Java server pages (JSP). It implements the MVC design pattern in the following way:

- the model and the system's state is implemented by business logic beans, i.e. java classes,

- the view is defined by means of Java server pages that generate html code for the client browser,

- the controller is implemented by servlets that receive requests (from the client browser), and dispatches control to the appropriate business control logic. Dispatching is performed on the grounds of a configuration file that defines how to map requests (URIs) to business logic beans that handle them, or to further Java server pages.

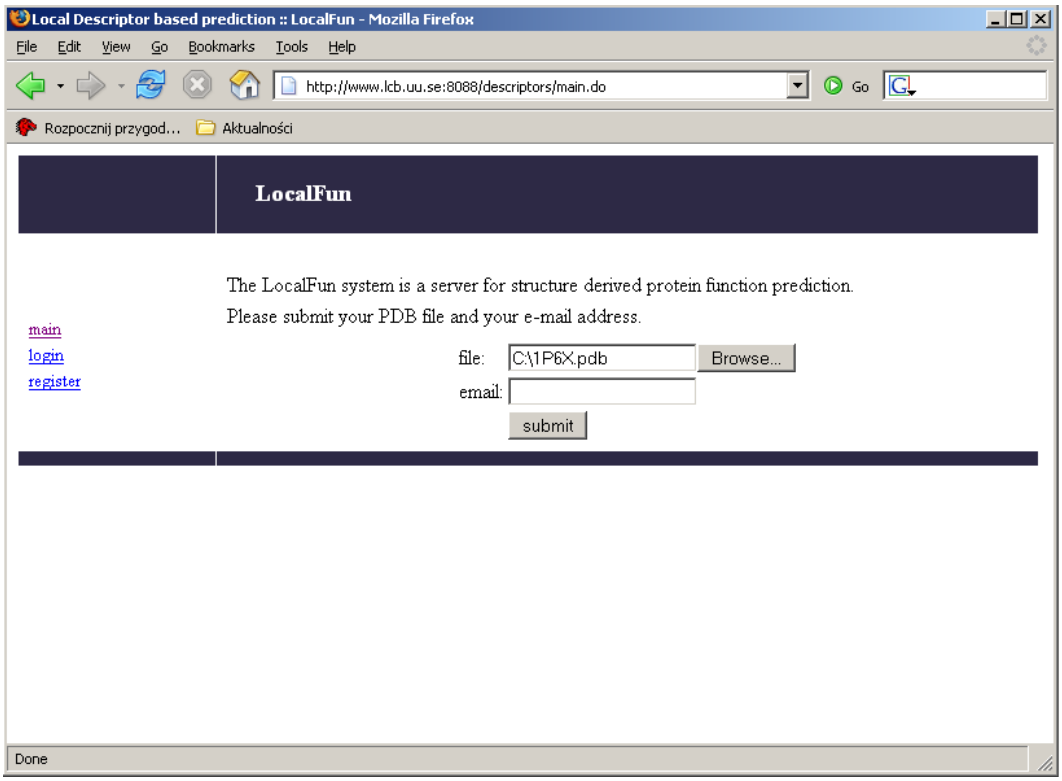

Figure 4.2. Web server

4.2.2 Request handling process

The process of handling a user request is as follows. The user submits their pdb file using the web browser interface (Figure 4.2). Apart from submitting the file, they have to provide their email address to which the results will be sent. Having submitted the file, they get a conformation that the task has started.

The servlet responsible for handling requests starts a new thread that will process the submitted file. The new thread first invokes a C++ program for extracting the descriptors and and assigning them to the existing clusters. The representative descriptors are written in a file, the program loads them and compares them to the descriptors found in the protein. The program results in a feature representation of the proteins, i.e. a vector of 1s and 0s. The vector is written to a text file that can be read by the Rosetta system that is run next. The resulting Rosetta files are sent to the user by email. An exemplary file with prediction result looks as follows:

```
PREDICTIONS

protein          prediction

1P6X             GO:0004674(5);GO:0004024(15);GO:0005524(20);
```

5 RESULTS

5.1 Descriptor data

All the analysis was performed on a representative subset of 657 proteins. There were 148,012 descriptors of at least 3 segments found. The mean size of a descriptor was ~ 27 residues. Figure 5.1 illustrates the number of segments distribution, showing that the number of descriptors decreases exponentially with the increase of the number of segments.

Grouping of the descriptors resulted in 146,682 nonempty preliminary groups. The average group size was 64, the biggest group of descriptor d1f60b_#53 from protein 1f60 contained 3990 descriptors. As one can expect, the descriptors of smaller size gather more group members, than the more specific, bigger descriptors. The most common conformation of descriptor d1f60b#53 consists of 3 segments and only 17 residues.

After the cleaning transformation 120,835 nonempty groups remained. The most popular group has 2831 members, the average group size is 39.5, therefore approximately 30% of preliminary group members were sieved out. The cleaned descriptors are smaller, on average a seed descriptor has 20 residues. The average $RMSD_{adj}$ score (see Equation 3.4) is 1.263 with the standard deviation of 0.606.

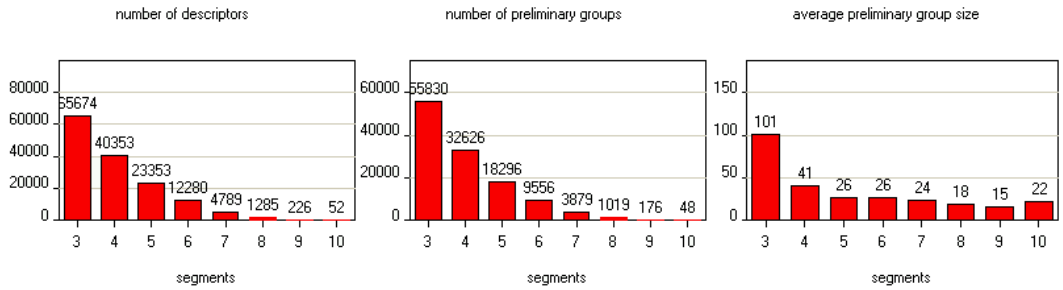

Figure 5.1. Left: Number of descriptors versus number of segments in descriptors. Middle: Number of segments of the preliminary group seed descriptor versus number of preliminary groups. Right: Number of segments of the preliminary group seed descriptor versus the mean size of preliminary group.

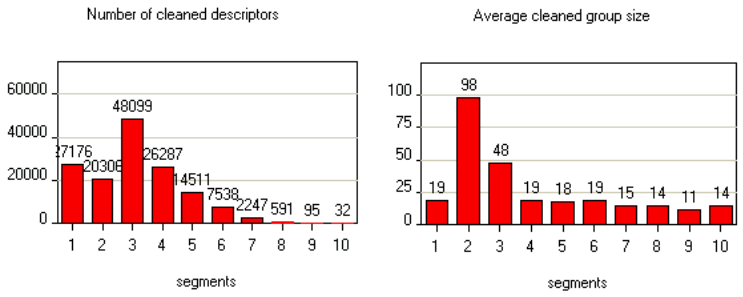

Figure 5.2. Statistics for the cleaned groups. Left: Number of segments versus number of cleaned descriptors. Right: Number of segments of the cleaned group seed descriptor versus the mean size of group.

5.2 Clustering

The descriptors were clustered using the two algorithms described. The hierarchical algorithm does not require any parameters, however the clusters depend on the level on which the dendrogram is cut.

The DBSCAN algorithm performance depends on the neighborhood radius ε and the minimum number of neighbors parameter $MinPts$. The authors [10] propose the following heuristic for choosing the ε parameter. For a given natural number k a distance from each point to its k-nearest point is computed. The points are sorted decreasingly according to those distances and plotted on a graph. The ε parameter should be set to the value of the k-distance of the point where the first "valley" starts, i.e. the points constitute a constant line (see Figure 5.3). The authors also suggest to set the k parameter value to 4. In various tests they performed the estmates obtained for the higher values of k did not significantly differ from the ones obtained for $k = 4$.

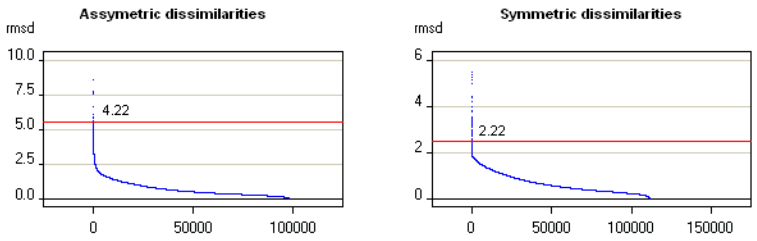

Figure 5.3. Setting the appropriate ε parameter value in DBSCAN algorithm. Left: $\varepsilon = 4.22$ for the asymmetric dissimilarity matrix; Right: $\varepsilon = 2.22$ for the symmetric dissimilarity matrix.

In [23] the groups of similar descriptors that assembled less than 7 members were disregarded. To gain a similar effect, the $MinPts$ parameter was set to 7 which leads to clusters of size at least 7. The ε values were 4.22 and 2.22 for the asymmetric and symmetric dissimilarities respectively.

5.2.1 Mutual information criterion

First, using the z-score figure of merit (Equation (3.12)) it was checked, whether the mutual information returned by the clusterings was significantly different than in random clustering. When computing the z-scores, it was assumed that the random clusters should have uniform sizes, which leads to maximization of the H_C entropy (3.10).

To examine whether the mutual information was an appropriate measure, the following test [13] was performed on data obtained from clustering descriptors with the DBSCAN algorithm, asymmetric dissimilarity matrix and parameters $\varepsilon = 4.22, MinPts = 7$. The original clustering combined $111,545$ descriptors into 3962 clusters, the remaining $35,337$ were assigned to the *noise* cluster. Beginning with the original clustering, two descriptors were repeatedly chosen at random and their cluster assignments were swapped. After each such swap the mutual information value was recomputed. In this fashion, the sizes of the clusters were constant, but the degree of correlation was slowly destroyed. The result,

shown in Figure 5.4, shows that adding noise to clusters decreases the mutual information.

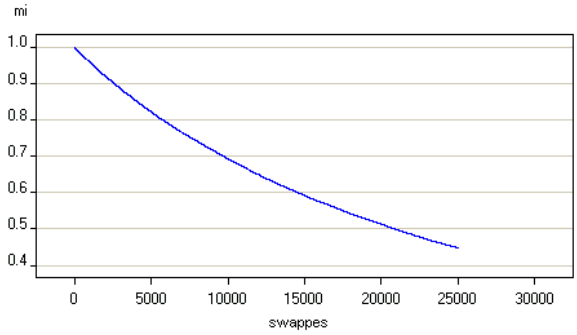

Figure 5.4. Number of cluster assignments swaps plotted versus the mutual information of the clustering.

The z-scores obtained by the algorithm indicate that the results are far from random clustering when the association with the Gene Ontology annotations is taken into account. The diagram in Figure 5.5 reports the z-scores obtained by the hierarchical algorithm plotted versus the number of clusters. Both the results for asymmetric and symmetric dissimilarities are presented. The z-scores obtained by the symmetric dissimilarities are slightly better. The DBSCAN algorithm resulted in significantly smaller number of clusters in both cases. All the small clusters were added to the *noise* cluster. The z-scores obtained were 650.473 and 727.156 respectively for the asymmetric and symmetric dissimilarities. The asymmetric case resulted in 3962 clusters, where $23,87\%$ of the data was assigned to the *noise* cluster, these descriptors were disregarded when computing the mutual information. The symmetric case resulted in 921 clusters with $22,5\%$ of the data in the *noise* cluster.

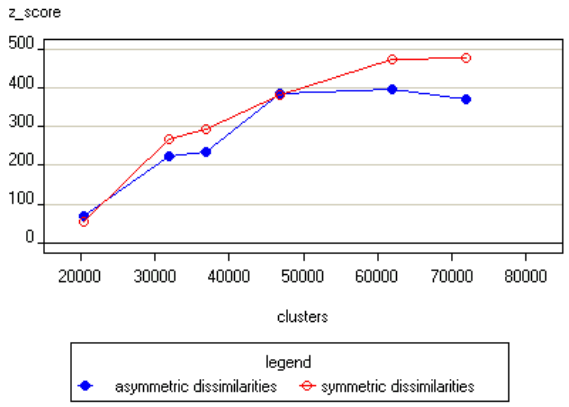

Figure 5.5. Number of clusters plotted versus z-score for the hierarchical algorithm; red - symmetric dissimilarity matrix, blue - asymmetric dissimilarity matrix.

Both algorithms obtained z-scores far from random. However it is hard to assess the appropriate parameters for training a classifier based on those results. In case of hierarchical clustering the z-scores obtained for a large number of clusters were higher then those for the smaller number of clusters. On the other hand the learning algorithms are known to lose accuracy for a large number of attributes, which might cause over-fitting.

5.3 Classifier performance

The classifier was trained on the decision tables provided by the clustering. 10-folds cross-validation was performed with ROC analysis returning the AUC value. The mean AUC values are compared in Table 5.1. Figure 5.6 compares the hierarchical algorithm performance plotted versus the number of clusters. The mean AUC values are reported for different algorithms and parameters settings.

Table 5.1. Classifiers

algorithm	symmetric	parameters	AUC
DBSCAN	no	$\varepsilon = 4.22, MinPts = 7$	**0.749421**
Hier. alg	no	20397 clusters	0.666366
Hier. alg	no	31882 clusters	0.716299
Hier. alg	no	36882 clusters	**0.741971**
Hier. alg	no	46882 clusters	0.728894
Hier. alg	no	61882 clusters	0.651937
Hier. alg	no	71882 clusters	0.60108
DBSCAN	yes	$\varepsilon = 2.22, MinPts = 7$	0.630853
Hier. alg	yes	20397 clusters	0.659613
Hier. alg	yes	31882 clusters	0.716299
Hier. alg	yes	36882 clusters	**0.738066**
Hier. alg	yes	46882 clusters	0.698626
Hier. alg	yes	61882 clusters	0.560662
Hier. alg	yes	71882 clusters	0.549193
Hier. alg	yes	96882 clusters	0.503305

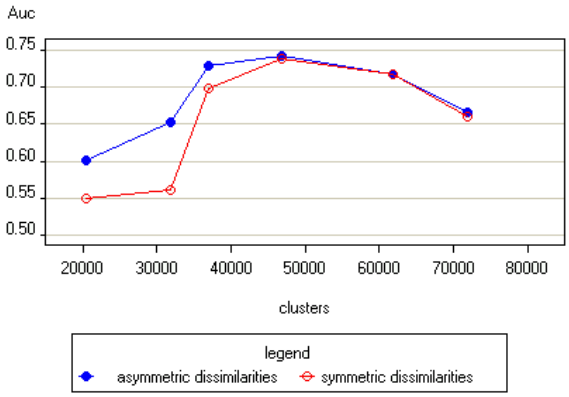

Figure 5.6. Number of clusters plotted versus mean AUC values for the hierarchical algorithm; red - symmetric dissimilarity matrix, blue - asymmetric dissimilarity matrix.

The highest AUC values were obtained for the unchanged, asymmetric dissimilarity matrix. The DBSCAN algorithm obtained AUC of 0.749421 whereas the hierarchical algorithm gained 0.741971. The DBSCAN algorithm gained very poor AUC value of 0.630853 for the symmetric dissimilarity matrix. This poor performance was probably caused by the uneven cluster sizes distribution. The majority of the descriptors, 94,004 out of 146,682 were assigned to the same cluster.

In order to assess the significance of the results, p-values were computed, i.e. the probabilities of the results obtained occurring by chance when the normal distribution of AUC is assumed. To this end the GO annotations from the original decision table were randomly

shuffled. The classifier was trained on this data a number of times in order to obtain a distribution of the AUC values.

Table 5.2 lists the AUC values obtained for each of the GO classes by the best classifiers. There are several classes that are very well predicted gaining AUC higher than 0.9. Assuming that the p-value smaller then 0.05 is significant, the results imply that the AUC value higher than 0.6 would not be obtained in a random model. For the DBSCAN algorithm the threshold was 0.6351, 0.642 for the asymmetric hierarchical algorithm and 0.6028 for the symmetric hierarchical algorithm.

Table 5.2. AUC values for GO classes obtained by the best clusterings. For each class reported is also the p-value for the AUC obtained.

GO	DBSCAN		Hier.alg. asymm		Hier.alg. symm		Number
	AUC	p-val	AUC	p-val	AUC	p-val	of proteins
GO:0008757	0.9166	0	0.8502	0	0.8495	0	50
GO:0004024	0.8834	0	0.9212	0	0.9702	0	19
GO:0004295	0.8822	0	0.7979	0	0.8754	0	62
GO:0020037	0.8762	0	0.897	0	0.8409	0	52
GO:0004263	0.8738	0	0.7851	0	0.8752	0	51
GO:0016765	0.8576	0	0.8295	0	0.7811	0	25
GO:0008135	0.829	0.0019	0.7797	0.0002	0.8106	0.0004	11
GO:0004674	0.802	0	0.7197	0	0.7082	0	64
GO:0005525	0.7955	0	0.7798	0	0.7845	0	77
GO:0017111	0.7821	0.0005	0.6447	0.0045	0.6224	0.0162	20
GO:0003964	0.7786	0.0068	0.8673	0	0.7763	0.001	10
GO:0004523	0.7786	0.0064	0.8673	0	0.7763	0.002	10
GO:0005125	0.7715	0.0015	0.642	0.0187	0.6084	0.0564	15
GO:0003779	0.7703	0	0.7782	0	0.7456	0	35
GO:0004896	0.7534	0.0159	0.7869	0	0.7894	0.0002	10
GO:0005509	0.7516	0	0.7216	0	0.7335	0	128
GO:0008083	0.7388	0.001	0.7278	0	0.7266	0	33
GO:0005524	0.7329	0	0.6991	0	0.7053	0	84
GO:0004842	0.7234	0.0004	0.6997	0	0.6489	0.0005	33
GO:0004867	0.7214	0.0057	0.5963	0.1189	0.6547	0.0424	11
GO:0004713	0.7117	0.0174	0.6075	0.0622	0.6828	0.0082	14
GO:0008270	0.6637	0	0.7315	0	0.671	0	171
GO:0003723	0.6392	0.0456	0.7279	0.0001	0.6781	0.0035	23
GO:0004222	0.6351	0.0352	0.7455	0.0001	0.7105	0.0012	21
GO:0003809	0.5895	0.2166	0.4925	0.4728	0.5729	0.1539	12
GO:0003700	0.4953	0.4813	0.7312	0.0002	0.6028	0.0485	20
GO:0004620	0.4814	0.4813	0.6053	0.0779	0.7269	0.0038	12

The classifiers were based on a number of IF-THEN rules. Below are presented some examples of the rules obtained:

- **IF** 1fi6a_#63 **AND** 1e4ua_#38 **THEN** function(GO:0005509)

  $Support = 20$

- **IF** 1fi6a_#58 **AND** 1fi6a_#63 **THEN** function(GO:0005509) **OR** function(GO:0005524)

  $Support = 20$

- **IF** 1dl5a1#169 **THEN** function(GO:0008757) **OR** function(GO:0004024) **OR** function(GO:0008270) **OR** function(GO:0003723)

$Support = 45$

The rules obtained were relatively short, the antecedent parts have up to two atomic descriptors, however the consequent parts often consist of more then one descriptor.

6 DISCUSSION

This thesis dealt with predicting protein function from structure. The method applied is based on a local protein descriptor idea. It was previously successfully applied by Hvidsten et al. [16], resulting in rule-based classifiers of high predictive ability.

The main contribution of the project is a software framework which leads to construction of a classifier for predicting protein function from structure. This involves finding local descriptors in proteins, comparing, grouping and clustering the descriptors. The software had to be designed to process very large amounts of data. A database was used for storing all the information, descriptors and their preliminary and cleaned groups, clusters of descriptors. I also managed to distribute the very time consuming process of comparing descriptors. It appeared to be very scalable, and the Swegrid clusters of 600 computers were used for computations.

The novel idea was to apply the clustering algorithms to the descriptor data. Many of the existing clustering algorithms are not suitable for processing such a big amount of data. Some algorithms need a number of clusters specified in advance which is very hard to determine in case of big datasets, especially when the data is given as pairwise dissimilarities between the objects. Both the clustering algorithms proposed, the density based DBSCAN algorithm and the single linkage version of the hierarchical algorithm were of relatively small complexity. The DBSCAN algorithm was $O(n\log(n))$, the hierarchical algorithm was $O(n^2)$ which might be much in case of big n, but the hierarchical algorithm has to be run only once as it does not require any parameters. Both algorithms resulted in clusters correlated with protein functions. The clusters were used for the information system construction.

The classifiers trained on the decision tables provided showed to have discriminatory ability, gaining mean AUC value higher then 0.74 with AUC higher than 0.9 for some classes. This proves that the method based the local protein substructures provides a powerful means of predicting protein function. The classifiers constructed were of a legible IF-THEN form, resulting in easily interpretable rules.

The classifier is made available through a web based server developed. Using an internet browser users can submit their proteins of interest and receive functions predicted by the classifier.

6.1 Future work

Applying the method to the whole set. The analysis in this thesis was performed only on a subset of ASTRAL version 1.67 domains. To assess the real classifiers evaluation the process should be repeated on the whole set. This task might be harder as it will involve more decision classes and the clustering algorithm will probably result in more clusters. On the other hand the number of objects will be about 10 times bigger (657 versus 6600 domains) so the size of the training set will increase.

Moreover, this process will be more computationally intensive. The comparison prob-

lem is $O(n^2)$, which means that adding new descriptors increases the time quadratically. Fortunately this stage is distributed on a computer grid.

The group cleaning step was not distributed despite the fact that each of the preliminary groups can be processed independently of the others. This was not necessary in case of the subset of domains the analysis was performed on. However the time complexity of $O(n \log(n))$ starts to be prohibitive in case of the big dataset. The solution for distributing this step on a computer grid should take into account the database usage.

According to the results obtained on a subset of proteins, the best AUC values gained by the hierarchical algorithm and the DBSCAN algorithm were comparable, both of about 0.74. However the hierarchical algorithm appeared to be more stable to the asymmetry of dissimilarities. The differences of the AUC values were very small there, with asymmetric matrix performing a bit better. Hence only this clustering method could be applied to the whole dataset.

Further development of the web server. The main focus in this thesis was on the off-line module of the system. So far, the web server provides the basic functionality for submitting protein structures and receiving prediction results by e-mail. The architecture of the on-line module is very flexible and easy to extend. The possible extensions may include:

- possibility for the user to specify to what chains of protein they want to apply the method,
- displaying the protein with full report concerning the parts of proteins matched by the representative descriptors,
- user account storage

Applying the method to other decision attributes. The whole process of preparing an information system, i.e. finding, comparing and clustering descriptors does not take the Gene Ontology classes into account. The proteins are annotated with the classes at the very last step. Hence any other type of information that can be associated with the structure could be used as a decision attribute.

Bibliography

[1] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. OPTICS: Ordering points to identify the clustering structure.

[2] Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., H, H. B., Cherry, J. M., Davis, A. P., Dolinski, K., Eppig, S. S. J. T., Harris, M. A., Hill, D., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J., Ringwald, M., Rubin, G., and Sherlock, G. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nature Genetics 25* (2000), 25–29.

[3] Bellman, R. E. *Adaptive Control Processes.* Princeton University Press, 1961.

[4] Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. The protein data bank. *Nucleic Acids Research 28* (2000), 235–242.

[5] Brenner, S. E. A tour of structural genomics. *Nature 2* (2001), 801–808.

[6] Breunig, M. M., Kriegel, H.-P., Ng, R., and Sander, J. LOF: Identifying density-based local outliers. In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD 2000)* (Dallas, 2000), TX, Ed., pp. 93–104.

[7] Brucker, P. On the complexity of clustering problems. *Lecture Notes in Economics and Mathematical Systems 157* (1978), 45Ű54.

[8] Chandonia, J., Nigel, G., Walker, S., Conte, L., Koehl, P., Levitt, M., and Brenner, S. E. The ASTRAL compendium in 2004. *Nucleic Acids Research 32* (2004), 189–192.

[9] Cover, T., and Thomas, J. *Elements of information theory.* D.L. Schilling, Wiley-Interscience,New York, 1991.

[10] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on Knowledge Discovery and Data Mining* (Portland, Oregon, 1996), E. Simoudis, J. Han, and U. Fayyad, Eds., AAAI Press, pp. 226–231.

[11] Frawley, W. J., Piatetsky-Shapiro, G., and Matheus, C. J. Knowledge discovery in databases - an overview. *AI Magazine 13* (1992), 57–70.

[12] Friedman, N., Geiger, D., and Goldszmidt, M. Bayesian network classifiers. *Machine Learning 29*, 2-3 (1997), 131–163.

[13] Gibbons, F. D., and Roth, F. P. Judging the quality of gene expression-based clustering methods using gene annotation. *Genome Research 12*, 10 (2002), 1574–1581.

[14] Hanley, J., and McNeil, B. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology 143* (1982), 29–36.

[15] Hastie, T., Tibshirani, R., and Friedman, J. *The elements of statistical learning: data mining, inference, and prediction: with 200 full-color illustrations.* Springer-Verlag Inc, 2001.

[16] Hvidsten, T. R. *Predicting Function of Genes and Proteins from Sequence, Structure and Expression Data.* PhD thesis, Uppsala University, 2004.

[17] Hvidsten, T. R., Kryshtafovych, A., Komorowski, J., and Fidelis, K. A novel approach to fold recognition using sequence-derived properties from sets of structurally similar local fragments of proteins. *Bioinformatics 19* (2003), 81–91.

[18] Jensen, L. J., Gupta, R., Staerfeldt, H. H., and Brunak, S. Prediction of human protein function according to gene ontology categories. *Bioinformatics 19*, 5 (2003), 635–642.

[19] Kabsch, W. *Acta Crystallographica A32* (1976), 922–923.

[20] Kaufman, L., and Rousseeuw, P. *Finding Groups in Data: an Introduction to Cluster Analysis.* John Wiley and Sons, 1990.

[21] Kearsley, S. K. On the orthogonal transformation used for structural comparisons. *Acta Crystallographica A45* (1989), 208–210.

[22] Komorowski, J., Øhrn, A., and Skowron, A. *Handbook of Data Mining and Knowledge Discovery.* Willi Klösgen and Jan M. Zytkow, 1975.

[23] Kryshtafovych, A., and Fidelis, K. Local descriptors of protein structure. I. general approach and classification of local 3d regions in proteins.

[24] McLachlan, A. D. *Acta Crystallogr A28* (1972), 656.

[25] Murzin, A., Brenner, S., Hubbard, T., and Chothia, C. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J Mol Biol 247(4)* (1995), 536–540.

[26] Ng, R. T., and Han, J. Efficient and effective clustering methods for spatial data mining. In *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings* (Los Altos, CA 94022, USA, 1994), J. Bocca, M. Jarke, and C. Zaniolo, Eds., Morgan Kaufmann Publishers, pp. 144–155.

[27] Pawlak, Z. Rough sets. *International Journal of Computer and Information Sciences 11* (1982), 341–356.

[28] Pitt, W. R., Williams, M. A., Steven, M., Sweeney, B., Bleasby, A. J., and Moss, D. S. The bioinformatics template library: generic components for biocomputing. *Bioinformatics 17*, 8 (2001), 729Ű737.

[29] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. *Numerical recipes 1st ed.* Cambridge University Press, Cambridge,UK., 1986.

[30] Sander, J., Ester, M., Kriegel, H.-P., and Xu, X. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery, an Int. Journal Kluwer Academic Publishers*.

[31] Shannon, C. E. A mathematical theory of communication. *Bell System Technical Journal*, 27 (1948), 379–423,623–656.

[32] Ślęzak, D. *Przybliżone redukty decyzyjne*. PhD thesis, Warsaw University, 2001.

Appendix A

Notation

AUC	area under ROC curve
$C\alpha$	α carbon atom
$C\beta$	β carbon atom
C	set of cleaned descriptors
D	set of descriptors
$Diss$	function measuring dissimilarity of two descriptors
$DissClust$	function measuring dissimilarity of two clusters
GO	Gene Ontology
$H(X)$	entropy of random variable X
$I(X,Y)$	mutual information of random variables X and Y
$RMSD$	root mean square distance
$RMSD_{adj}$	RMSD score for two descriptors normalized by the number of residues
ROC	Receiver Operating Characteristic

Appendix B

Diagrams

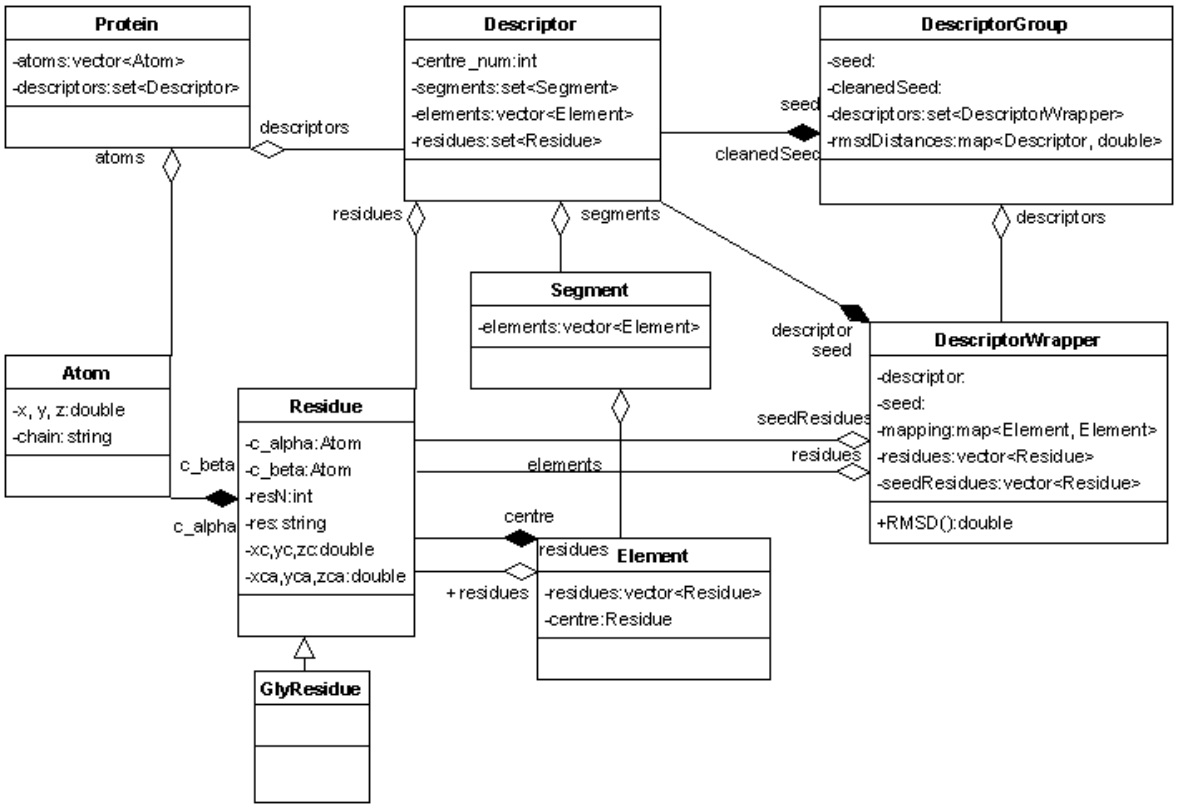

Figure B.1. Diagram of classes for data storage

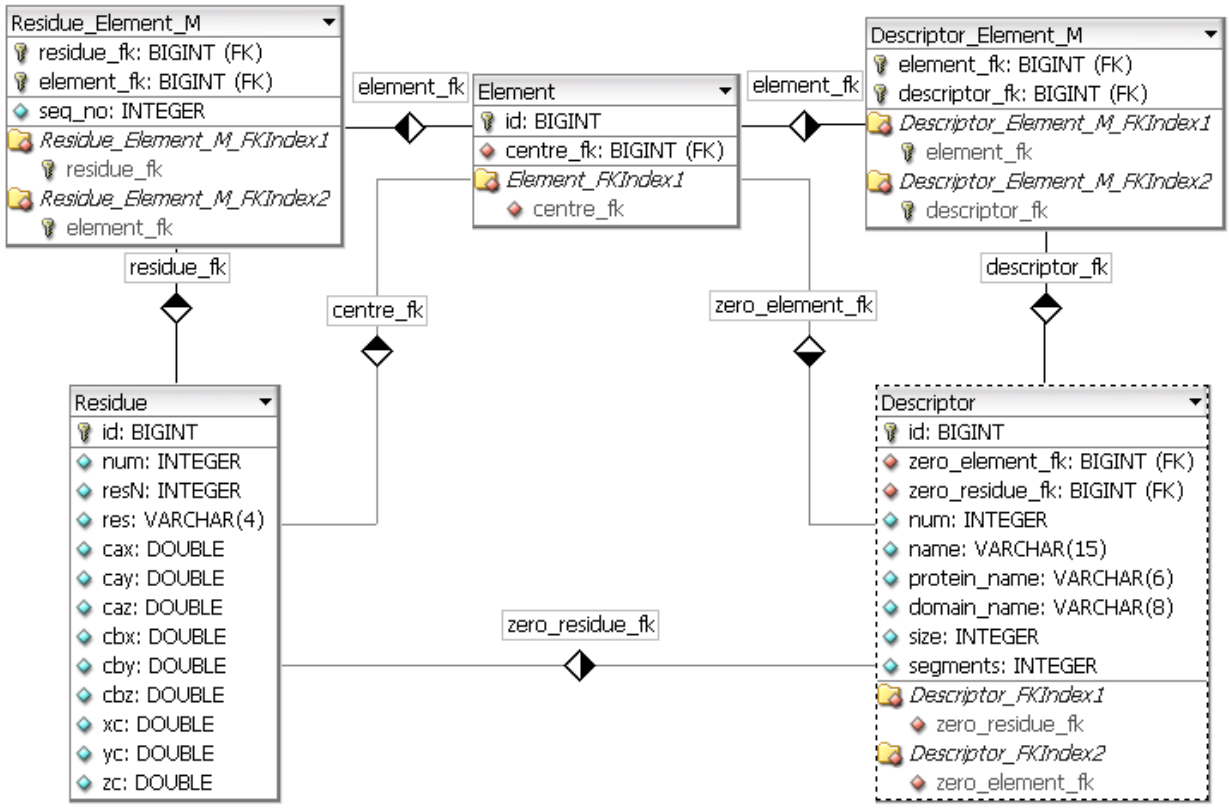

Figure B.2. Diagram of tables for representing descriptors.

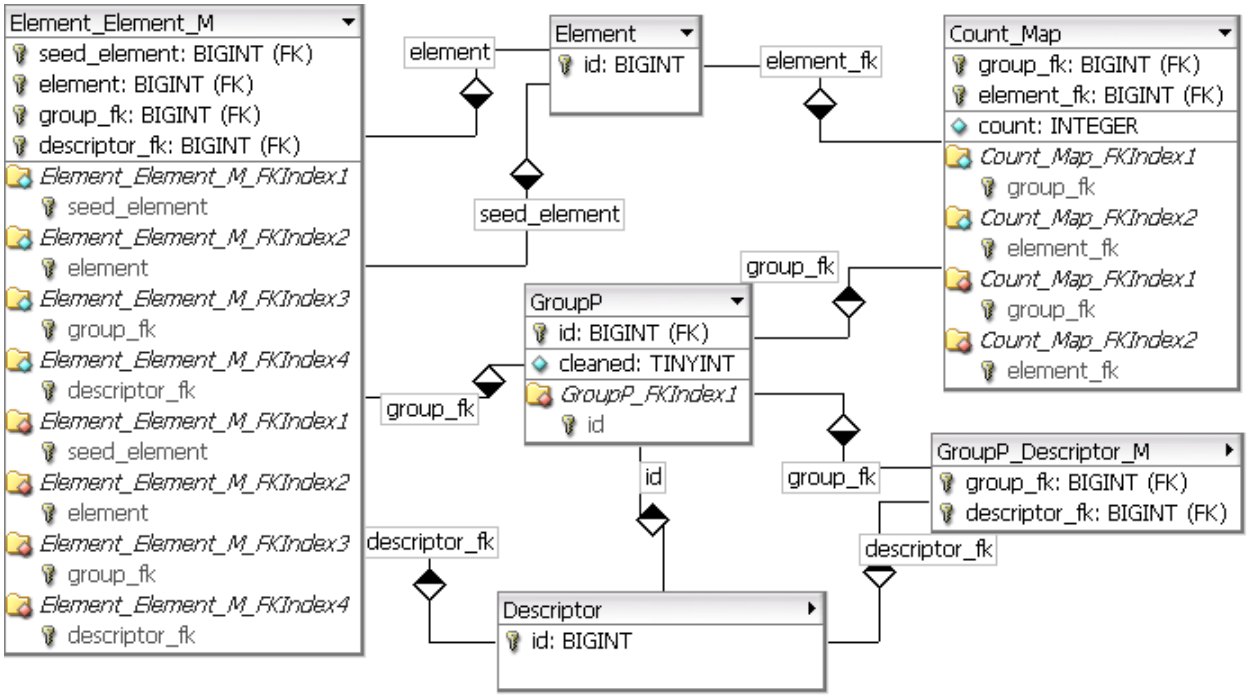

Figure B.3. Diagram of tables for storing preliminary groups information.

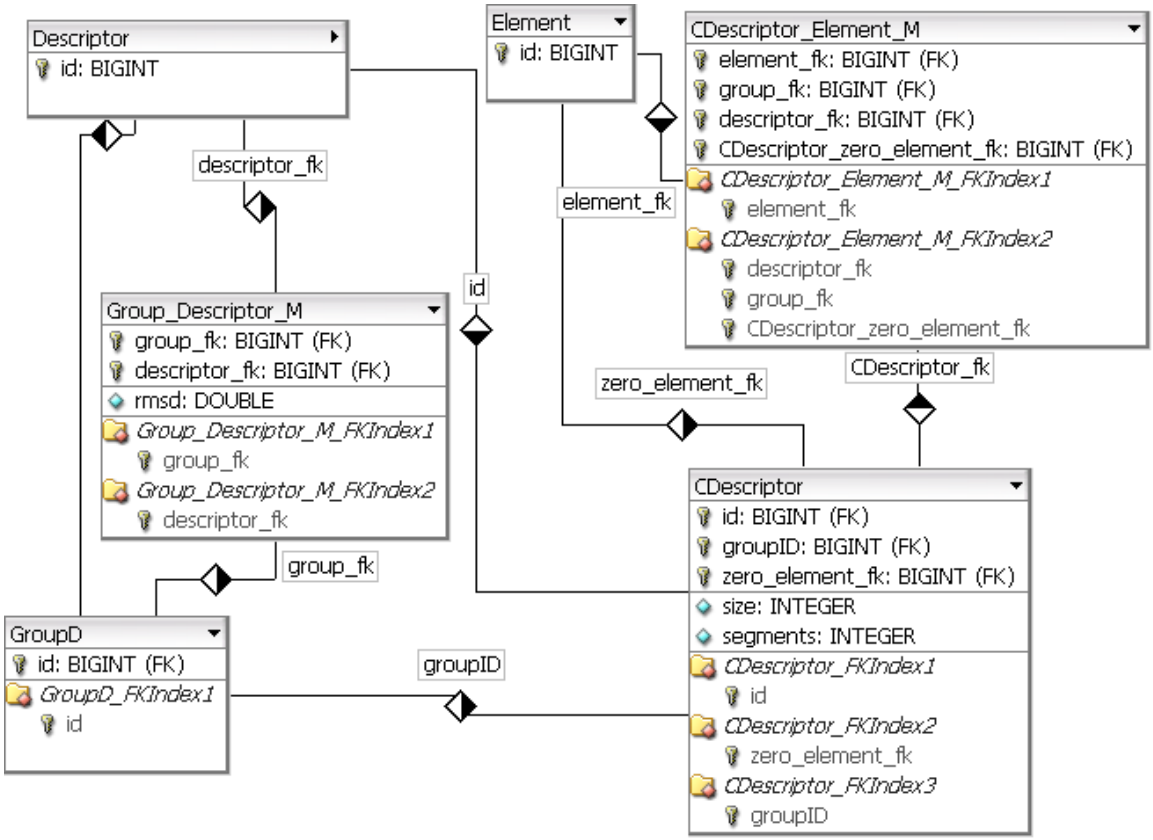

Figure B.4. Diagram of tables for storing cleaned groups information.

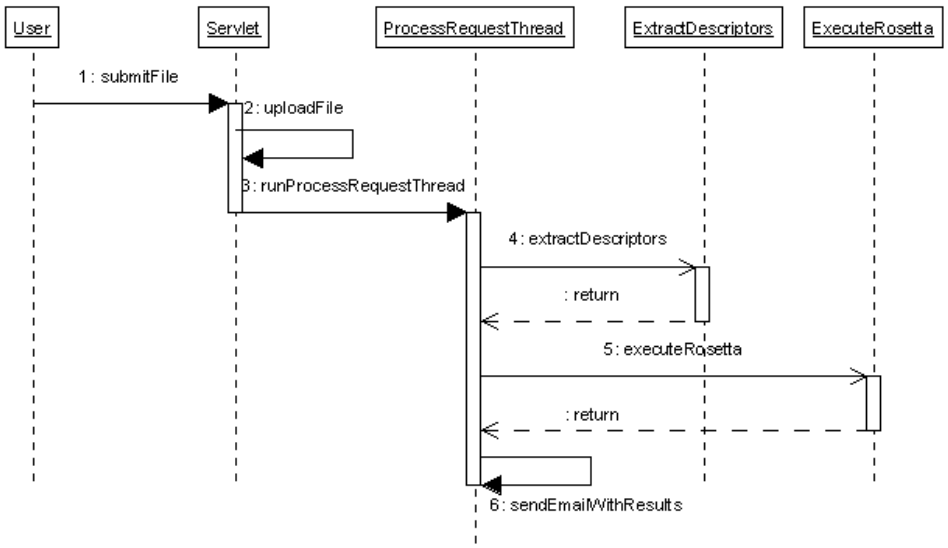

Figure B.5. An invocation diagram illustrating the request handling process of the web server.