Soft Comput (2011) 15:1897–1908
DOI 10.1007/s00500-010-0624-9

A genetic programming method for protein motif discovery and protein classification

Denise Fukumi Tsunoda · Alex Alves Freitas · Heitor Silvério Lopes

Published online: 16 June 2010

Abstract Proteins can be grouped into families according to some features such as hydrophobicity, composition or structure, aiming to establish common biological functions. This paper presents MAHATMA—memetic algorithm-based highly adapted tool for motif ascertainment—a system that was conceived to discover features (particular sequences of amino acids, or motifs) that occur very often in proteins of a given family but rarely occur in proteins of other families. These features can be used for the classification of unknown proteins, that is, to predict their function by analyzing their primary structure. Experiments were done with a set of enzymes extracted from the Protein Data Bank. The heuristic method used was based on genetic programming using operators specially tailored for the target problem. The final performance was measured using sensitivity, specificity and hit rate. The best results obtained for the enzyme dataset suggest that the proposed evolutionary computation method is effective in finding predictive features (motifs) for protein classification.

D. F. Tsunoda (✉)
Federal University of Parana, Av. Prefeito Lothário Meissner,
632, Room 38, Curitiba, PR, Brazil
e-mail: dtsunoda@ufpr.br

A. A. Freitas
School of Computing, University of Kent, Room S107,
Canterbury, Kent CT2 7NF, UK
e-mail: A.A.Freitas@kent.ac.uk

H. S. Lopes
Federal University of Technology, Av. 7 de Setembro,
3165, Bloco D, 3° floor, Curitiba, PR, Brazil
e-mail: hslopes@utfpr.edu.br

Keywords Evolutionary algorithms · Genetic programming · Data mining · Proteins patterns discovery

1 Introduction

This paper proposes a computational tool based on a genetic programming method, a type of evolutionary algorithm, specially devised for the automatic discovery of protein motifs (“signatures” or “patterns” characterizing proteins), using as input the primary structure (see below) of proteins.

Proteins are very large molecules responsible for several functions in living organisms, such as: transport of small molecules, sustentation, regulation, increase of reaction speed and others. Biological organisms have thousands of different types of proteins, which are constituted basically of amino acids linked in linear chains through peptide connections. The amino acid sequence of a protein’s polypeptide chain, also called primary structure, is inextricably linked to its function (Lehninger et al. 1998). Some different regions of the sequence form secondary structures like alpha (α) helices or beta (β) strands. The tertiary structure is formed by packing some structural elements into one or several compact units called domains. The final configuration of the protein also may contain several polypeptide chains arranged in a quaternary structure. Active intra-molecular forces like covalent peptide bonds and disulfide bonds cause proteins to assume specific three-dimensional shapes that are directly related to their biological functions (Branden and Tooze 1999). Proteins are grouped into super families, families and subfamilies according to these biological functions (Friedberg 2006; Rost et al. 2003; Jensen et al. 2002).

For instance, according to Lehninger et al. (1998), proteins can be categorized into broad groups such as enzymes (highly specialized proteins with catalytic activity, e.g., catalase, that catalyzes the decomposition of hydrogen peroxide into water and oxygen), transport proteins (which in blood plasma bind and carry molecules or ions from one organ to another, e.g., hemoglobin, which transports oxygen), storage proteins (required for the growth of the germinating seedling for some plants, e.g., ovalbumin, casein and ferritin), motile proteins (which endow cells and organisms with the ability to contract, change shape or move, e.g., myosin and actin), structural proteins (which serve as supporting structures, offering strength or protection, e.g., collagen and keratin), defense proteins (which defend organisms against invasions by other species or protect them from injury, e.g., immunoglobulin and fibrinogen), regulatory proteins (which help to regulate cellular or physiological activity, e.g., insulin) and others.

Genome-sequencing technology has produced a huge amount of data about proteins and their primary structure (amino acid sequence). However, there are a large number of proteins whose function is unknown. Hence, an active research area consists of predicting proteins' functions based on proteins' primary sequences. Despite the existence of several methods to solve this kind of protein function prediction problem (Chua et al. 2006; Zhao et al. 2008), it still remains one of the main challenges in the current post-genomic era.

Evolutionary algorithms are search and optimisation methods inspired by the principle of natural selection in biological evolution. In essence, they evolve a population of candidate solutions ("individuals") to a target problem, doing a search in the space of candidate solutions guided by a "fitness function", which measures the quality of candidate solutions. In general the higher the fitness of individuals, the more likely they are to be selected to reproduce, creating new candidate solutions that inherit characteristics of their "parent" individuals. Hence, the population gradually evolves to better and better candidate solutions as measured by the fitness function. Genetic programming is a particular type of evolutionary algorithms where an individual (candidate solution) consists not only of data (variables or constants), but also of operators (or functions) applied to the individual's data. Hence, in genetic programming individuals can be said to represent "programs", in a loose sense, or "executable structures".

The proposed genetic programming method—called MAHATMA (memetic algorithm-based highly adapted tool for motif ascertainment)—finds subsequences of amino acids (patterns, features or motifs expressions) that occur very often in proteins of a given class (family) but rarely occur in proteins of other classes. Those discovered motifs can be further used for the characterization of families of proteins as well as for the automatic classification of unknown-class proteins.

The remainder of this paper is organized as follows. Section 2 presents algorithmic details of the proposed genetic programming method, like individual representation, selection method, genetic operators, and others. Sections 3 and 4 present the setup of the computational experiments and their results, respectively. In these sections parameter tuning experiments and results are reported in order to decide which components of the method to use and to explain their influence in the performance of the method. Section 5 reviews related work. Finally, conclusions and future research directions are provided in Sect. 6. This paper is an extended version of Tsunoda et al. (2009).

2 The proposed genetic programming method

Eiben and Smith (2003) state that there are many variants of evolutionary algorithms (EAs) with a common basic idea: given an initial population of individuals, the environmental pressure causes natural selection (survival of the fittest) and hence the improvement (a rise in the fitness) of the population.

The genetic algorithm (GA) proposed by John Holland at the University of Michigan is a subclass of evolutionary algorithms (EAs) that has proven to be successful in solving some difficult problems (Goldberg 1989). GAs are based on the mechanics of natural selection, in others words, inspired from the Darwinian theory of evolution.

Genetic programming (GP) (Koza 1992, 1994) has been used mainly for its ability to perform adaptive and robust searches. Besides, as an evolutionary computation technique, it operates in parallel over a population of candidate solutions, allowing a simultaneous exploration of different regions of the search space in the solution domain. This characterizes a global search, less likely to get trapped in local optima, by comparison with many local-search methods.

The combination of evolutionary algorithms with local-search operators that work within the EA loop has been termed "memetic algorithms" (Moscato 1989). This term also applies to EAs that use operators based on problem-specific knowledge. According to Eiben and Smith (2003), memetic algorithms (MA) or hybrid algorithms (HA) have been shown to be orders of magnitude faster and more accurate than EAs on some problems, and are the "state of the art" on many search or optimisation problems.

2.1 Basic algorithm and individual representation

MAHATMA is a hybrid genetic programming (GP) based tool (Koza 1992, 1994; Hsu 2009). In GP, like in other types of evolutionary algorithms, each individual corresponds to a candidate solution to the target problem. As mentioned in Sect. 1, the key characteristic that distinguishes GP from other evolutionary algorithms is that the former evolves candidate solutions representing "executable structures", consisting of both data and operators (functions); whilst in other types of evolutionary algorithms such as GAs a candidate solution typically consists of data only (and not operators/functions).

In this work the goal of the GP method is to find a set of rules combining protein motifs which, when used as predictive features by a classification algorithm, lead to a high protein-classification accuracy. In this work, an individual is represented by a tree (Fig. 1). There are three kinds of nodes: root node, intermediate nodes and leaf nodes. The root and intermediate nodes represent the logical operations: *and*, *or* and *not*. The leaf nodes are variable-length sequences of amino acids representing candidate protein motifs.

Hence, each individual represents the antecedent (IF part) of an IF–THEN classification rule consisting of a motif formed by applying logical operations to the names of amino acids in the proteins' amino acid sequences. Each amino acid name can be abbreviated by a single letter. For instance, the individual shown in Fig. 1 can be read as the rule antecedent: IF "(a protein has the aminoacid subsequence MD or MM) and (a protein has the aminoacid subsequences LQE and IGA)".

Table 1 presents the 20 "standard" monomeric units of proteins, the amino acids (Lesk 2001). It is from these substances that proteins are synthesized.

The class predicted by the THEN part of a rule is computed by using a deterministic procedure that assigns the best possible class to the rule (individual), to be explained later.

Figure 2 presents MAHATMA's flowchart. The steps of this flowchart will be explained in the following subsections.

2.2 Selection method and genetic operators

The system uses stochastic tournament selection, which works as follows (Banzhaf et al. 1998). First, k individuals are randomly drawn from the current population, with replacement, where k is determined as a percentage of the population size. In this work, k is 3% of the population size (this is a user-defined parameter). Then, the k individuals are prompted to "play a tournament", where the probability of an individual to win the tournament is proportional to its fitness value. A copy of the winner of a tournament is then passed on, as a parent, to genetic operators such as crossover and mutation. Notice that each tournament selects just one parent, so that the tournament selection procedure has to be called N times to produce N parents, where N is the population size. The choice of k must be done carefully, since this parameter modulates the degree of the selective pressure. The larger the k, the higher the selective pressure will be, possibly leading the algorithm to converge rapidly into a "local maximum". On the other hand, a k too small will impose no selective pressure, turning the method into a random search.

We emphasize that MAHATMA has two kinds of operators: structural operators (usual in GP Koza 1992, 1994) and leaf operators (based on genetic algorithms Goldberg 1989; Larose 2006). The structural operators are: reproduction, crossover, mutation and editing. The reproduction operator just copies a selected individual to the next generation.

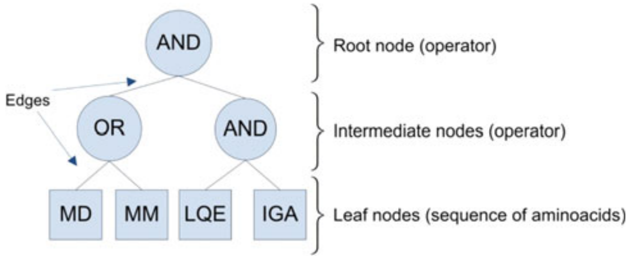
AND
Edges
OR
AND
MD
MM
LQE
IGA
Root node (operator)
Intermediate nodes (operator)
Leaf nodes (sequence of aminoacids)

Fig. 1 MAHATMA individual representation

Table 1 The amino acids and their three-letter and one-letter codes

Amino acid	Three-letter symbol	One-letter symbol[a]
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamic acid	Glu	E
Glutamine	Gln	Q
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

[a] The one-letter symbol for an undetermined or "nonstandard" amino acid is X

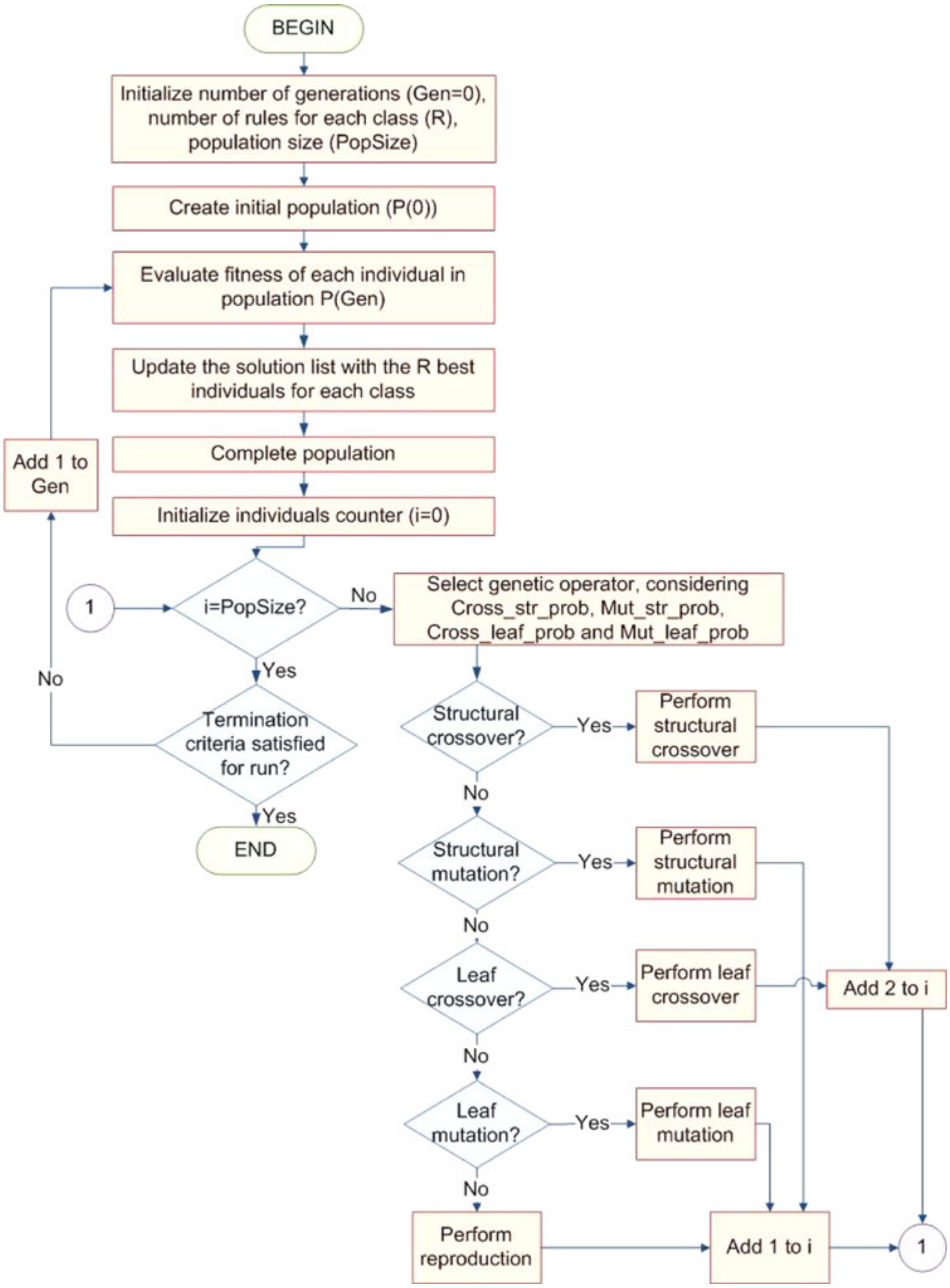
BEGIN
Initialize number of generations (Gen=0), number of rules for each class (R), population size (PopSize)
Create initial population (P(0))
Evaluate fitness of each individual in population P(Gen)
Update the solution list with the R best individuals for each class
Complete population
Initialize individuals counter (i=0)
Add 1 to Gen
1
i=PopSize?
No
Yes
Select genetic operator, considering Cross_str_prob, Mut_str_prob, Cross_leaf_prob and Mut_leaf_prob
No
Termination criteria satisfied for run?
Yes
END
Structural crossover?
Yes
Perform structural crossover
No
Structural mutation?
Yes
Perform structural mutation
No
Leaf crossover?
Yes
Perform leaf crossover
Add 2 to i
No
Leaf mutation?
Yes
Perform leaf mutation
No
Perform reproduction
Add 1 to i
1

Fig. 2 MAHATMA flowchart

The leaf operators modify the sequence of amino acids by genetic operators (e.g., crossover and mutation) in order to produce offspring (Goldberg 1989; Larose 2006).

2.2.1 *Structural operators*

These operators modify an individual's structure. MAHATMA's structural mutation introduces random changes in structures. For example, in the "parent" individual in the left part of Fig. 3, the AND at the intermediate node is selected as the mutation point. A subtree is randomly generated and inserted at that point, to produce the "child" individual.

The structural crossover operator produces new offspring taking parts from each of the two parents. It is also called sexual recombination. For example, in the two "parent" individuals in the left part of Fig. 4, one random point in each parent is select. Each of these points is a rooted subtree crossover point. The right part of Fig. 4 shows the two offspring resulting from crossover.

The edition operation provides a means to edit and simplify expressions as genetic programming is running.

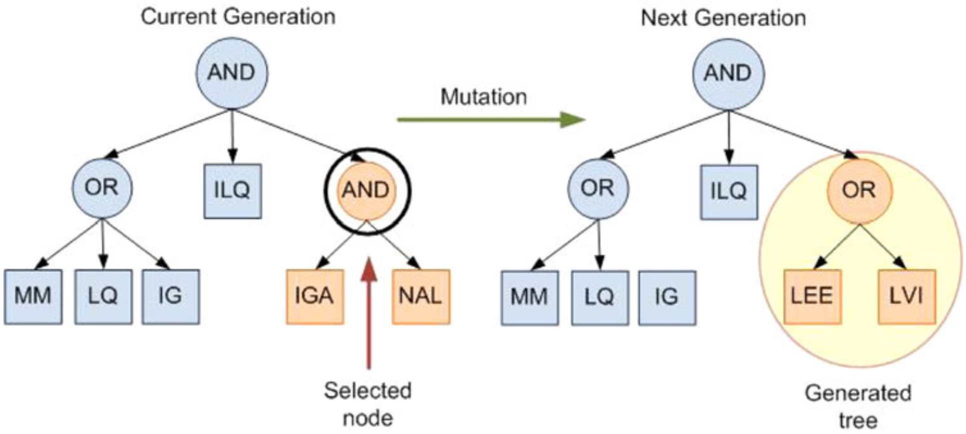

Fig. 3 MAHATMA structural mutation operator

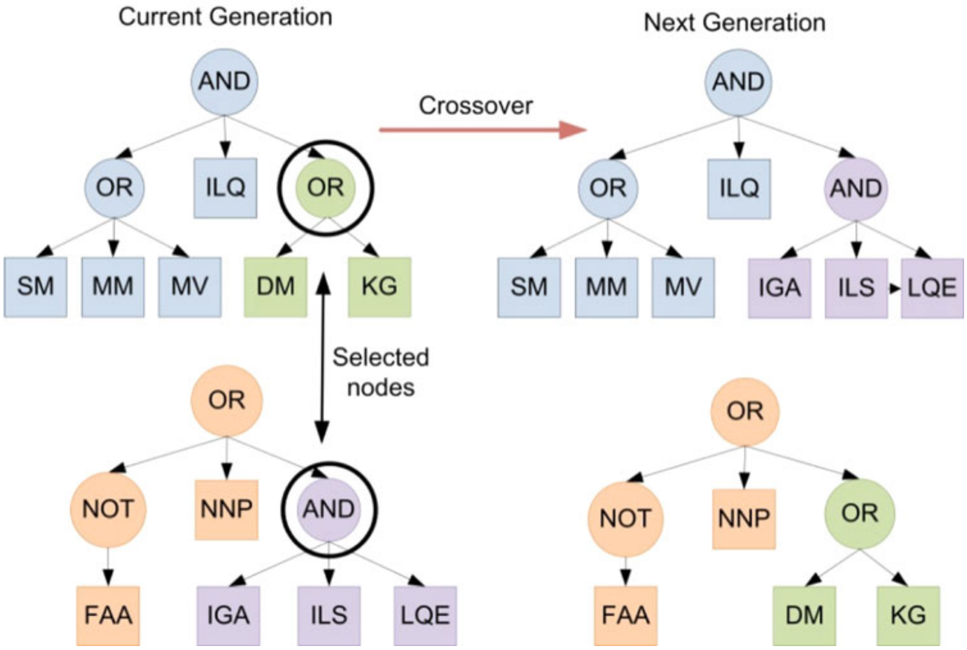

Fig. 4 MAHATMA structural crossover operator

Table 2 Editing rules

Before operation	After operation
X or X	X
A and A	A
Not not B	B
(C and C) or C	C
D and (D or D)	D

Edition is an asexual operator and it recursively applies a set of simplifying operations (*editing rules*, Table 2) in order to optimize the rule. If any function has no side effects, the edition operator will evaluate that function and replace it with the value obtained by the evaluation. Figure 5 shows an expression before and after the MAHATMA's editing operation.

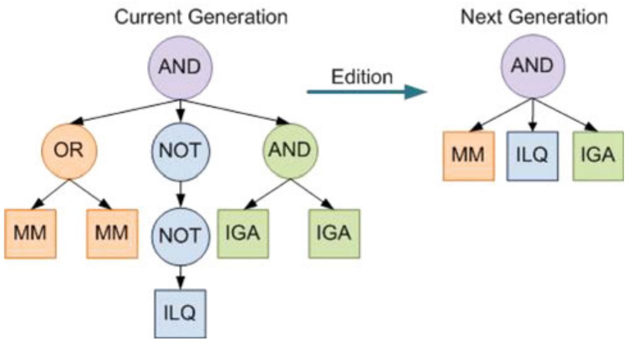

Fig. 5 MAHATMA edition operator

2.2.2 *Leaf operators*

These operators modify the contents of leaf nodes (sequences of amino acids representing motifs). MAHATMA uses the classical one-point crossover often used in GAs, where a crossover point is randomly selected and then the two parents swap their genetic material from the crossover point up to the right-hand end of the individual (Hsu 2009). Notice, however, that this kind of crossover was originally designed for a fixed-length individual representation, unlike the variable-length motif representation used in this work. Therefore, this work has adapted the conventional one-point crossover to a variable-length representation, as follows. The crossover point (which is still randomly generated) indicates the percentage of the genome of each parent where the swapping of genes starts. The percentile (relative position) is the same for both parents, but the actual (absolute) position where the gene swapping starts can be different, since the parents can have different numbers of genes. This is illustrated in Fig. 6, where the crossover percentage is 60%. The absolute position of the crossover point for each parent is computed by multiplying 0.6 by the number of genes of the parent and rounding up the result. This results in crossover points at positions 4 and 5 in the first and second parents, respectively. The genetic material being swapped is shown in Fig. 6.

The crossover operator introduced here also has another feature that distinguishes it from conventional crossover operators. This feature consists of monotonically increasing the fitness of the children with respect to their parents, and it was introduced to eliminate the potentially destructive effect of crossover (which can produce offspring with fitness worser than the parents). This idea works as follows. After crossover has been done, all the corresponding four individuals (two parents and two children) are compared to each other and the best two individuals are passed to the next population, no matter whether the individuals being passed are parent or offspring.

This work introduces four kinds of mutation operators tailored for the variable-length sequence of amino acids represented by each individual, as follows:

a. *Addition to the left* (*AE*) a letter, representing an amino acid, is randomly generated and inserted into the leftmost end of the sequence of amino acids;
b. *Addition to the right* (*AR*) analogous to AE, with the difference that the new amino acid is inserted into the rightmost end of the sequence of amino acids;
c. *Multiple mutations* (*MM*) each of the amino acids from a randomly generated starting position up to the end of the sequence is replaced by another randomly generated amino acid. The starting position can be any position in the sequence except the first and the last positions.
d. *Removal* (*RM*) the amino acid in a randomly chosen position is removed from the sequence. Notice that after removal of an amino acid the sequence will still have at least three amino acids. If this condition is not met then this operator is not applied, and another mutation operator is applied instead.

These mutation operators also have the feature of monotonically increasing the fitness of offspring with respect to the parents, as explained for the crossover operator. That is, if the fitness of the offspring is worser than the fitness of the parent then the offspring is thrown away and the parent is passed to the next generation.

The system also has an extra genetic operator designed specifically for the target problem. This operator, called the expansion operator, performs a kind of local search in the solution space, so that the MAHATMA can be considered a hybrid method or a memetic algorithm (Moscato 1989). The expansion operator works as follows.

The basic idea is to increase the length of the motif represented by an individual, making that motif more specific to a given class, while at the same time increasing the motif's ability to discriminate between different classes of proteins. The operator starts by randomly selecting a protein among those that contain the motif represented by the individual to be expanded. (If there is no protein with that motif, the operator is not applied.) The selected protein is then used as a source of amino acids to be inserted into the individual, as follows.

First, the amino acid which is located immediately to the left of the motif in the protein is inserted into the leftmost end of the individual's sequence of amino acids, and the

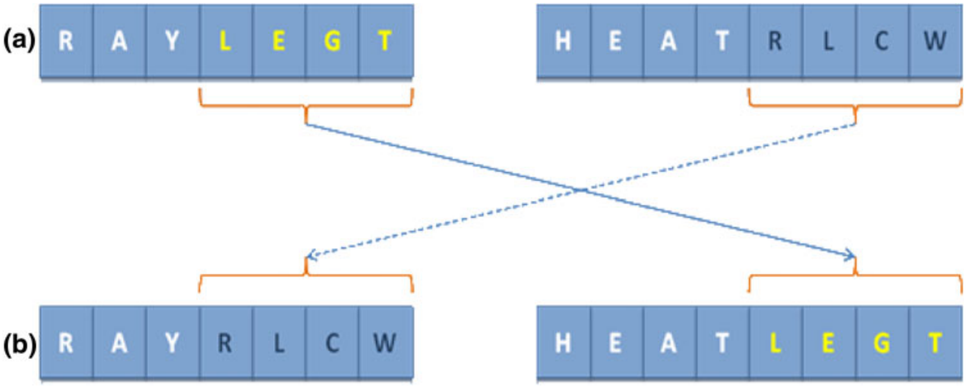

Fig. 6 One-point crossover between variable-length parents: **a** original parents, **b** offspring

individual's fitness is recomputed. If the new fitness is worser than the previous one, then this operation is undone—i.e., the just-added amino acid is removed from the individual's sequence of amino acids—and the expansion based on the current protein is terminated. Otherwise the just-inserted amino acid is kept in the individual, and the process continues. Next, the amino acid which is located immediately to the right of the motif in the protein is inserted into the rightmost end of the individual's sequence of amino acids, and the fitness of the individual is recomputed. Again, if the new fitness is worser than the previous one, this operation is undone and the expansion based on the current protein is terminated. Otherwise the just-inserted amino acid is kept in the individual, and the process continues. This process is repeated, considering amino acids that are 2, 3, …, positions away from the motif in the current protein, alternating between amino acids to the left and to the right of that motif, until an attempt to further expand the individual would lead to a reduction in its fitness. The basic idea of this expansion operator is conceptually similar, at a high level of abstraction, to the well-known BLAST method for protein alignment, which was presented by Altschul et al. (1990). This method implements a dynamic programming approach to protein alignment, a technique to find motifs in a set of proteins. Nisbet et al. (2009) state that Blast will take a sequence and compare it to a database of already-analyzed sequences to check for matches and thus help in identifying a DNA, RNA or protein sequence. Blast is available at the NCBI website.[1] Note that BLAST is not typically used for classification, whilst the proposed expansion operator works in the context of a genetic programming algorithm for classification.

Next, this process is repeated for all other proteins that also contain the individual's motif and that belong to the same class as the class of the protein that was used in the first step of the operator.

Hence, the expansion operator aims at generating the longest (most specific) motif for a given class, but notice that the expansion process never decreases the fitness of the individual being expanded. Therefore, this operator also has the feature of monotonically increasing the fitness of the offspring with respect to its parent, like the crossover and mutation leaf operators.

2.3 Fitness function

As mentioned earlier, an individual represents a protein motif that will be used as a predictor attribute by a given classification algorithm. Since the goal is to maximize classification accuracy, the quality of a motif is determined by its ability in discriminating proteins of different classes. That is, ideally a motif should represent an amino acid sequence that occurs in many proteins of a given class and in no (or few) proteins of other classes. The fitness function was designed to take this basic principle into account. Hence, the fitness of an individual (motif) is computed as follows.

At first, MAHATMA computes, for each class i, $i = 1$, …, 6 (for the enzyme dataset used in this work), the relative frequency of occurrence of the motif in that class. This is simply the number of proteins of the ith class where the motif occurs in the protein's primary sequence. Secondly, it computes, for each class i, a measure of the ability of the motif to discriminate between class i and the other classes, denoted $Disc_i$ and given by Eq. 1, where F_i is the relative frequency of the individual's motif in the ith class, n is the number of classes ($n = 6$ in this work), and k is the number of classes that contain at least one protein whose primary sequence contains the individual's motif. Hence, the fitness function can be formally stated as follows:

$$Disc_i = F_i \times \left[1 - \sum_{j=1}^{n}\left(\frac{F_{j,j\neq i}}{(k-1)}\right)\right] \tag{1}$$

where the rightmost term of the formula simply computes the average relative frequency of the motif in all the $(n - 1)$ classes j with $j \neq i$. This term is subtracted from 1, so that the term between square brackets is to be maximized; the higher its value, the better the value of $Disc_i$. Similarly, the value of F_i (the first term of the formula) is also to be maximized, so that a high value of $Disc_i$ means that the motif occurs very often in class i but rarely in the other classes.

Finally, once the value of $Disc_i$ has been calculated for all classes i, $i = 1$, …, n, the motif is associated with the class i that has the largest value of $Disc_i$, and that value is considered the fitness of the individual.

Hence, the motif is considered as a characteristic pattern of proteins belonging to class i. In other words, the occurrence of that motif in a protein of unknown class will be considered, by the classification algorithm, as evidence that the protein belongs to class i.

2.4 Result designation

As explained earlier, each individual represents a motif which is associated with a given class of proteins. Therefore, it is not enough to return, as solution found by the method, only the best motif found throughout the evolutionary process; that is usual in conventional evolutionary algorithms. It is necessary, in this kind of problem, to return a set of motifs, in order to perform a comprehensive classification of proteins into known families. In this work,

[1] Available at http://www.ncbi.nlm.nih.gov/blast.

we return the best M motifs found throughout the evolutionary process, where M is a user-defined parameter.

3 Setup of the computational experiments

The data set to be mined consists of data about enzymes. The data was extracted from the PDB (Protein Data Bank), version 102, by identifying the PDB entries which had an EC number. The PDB is "the single worldwide repository of information about the 3D structures of large biological molecules, including proteins and nucleic acids" (RCBS 2010). It is freely available at the RCBS website.[2] The EC number is an enzyme code provided by the International Union of Biochemistry and Molecular Biology (IUBMB). From a data mining viewpoint, each EC number corresponds to a class, i.e., a specific protein function. More precisely, the EC number consists of four digits, where each pair of adjacent digits is separated by a dot ("."), and it specifies the chemical reaction catalyzed by the corresponding enzyme. For instance, the enzyme *alcohol dehydrogenase* has the number EC.1.1.1.1.

Note that this is a hierarchical classification consisting of four levels, so that the first digit represents the most general classes and the last digit the most specific subclasses. In this work we address the prediction of the first digit only, corresponding to the prediction of the most general class to which the example belongs. We emphasize that this is still a useful, challenging prediction, and other projects have also focused on the prediction of the first digit only, see e.g., (desJardins et al. 1997); (Izrailev and Farnum 2004) and (Santos et al. 2009). For a review of hierarchical classification applied to protein function prediction, see (Freitas and de Carvalho 2007); and for a recent review of hierarchical classification in general, see (Silla Jr and Freitas 2010). The first digit can take on six different values, corresponding to the following six different classes: EC.1, oxidoreductases; EC.2, transferases; EC.3, hydrolases; EC.4, lyases; EC.5, isomerases and EC.6, ligases.

Some of the enzymes stored in the PDB contained nonstandard amino acids, from which no useful motif can be discovered. Therefore, as part of our data preparation procedure, we have only retrieved from PDB the enzymes whose primary sequence has at least 30 standard amino acids. After this simple filtering, the total number of proteins retrieved from the PDB was 8,399, distributed across the six classes as follows: 1,583 proteins in class EC.1; 1,866 in class EC.2; 3,385 in class EC.3; 775 in class EC.4; 481 in class EC.5, and 309 in class EC.6.

As described earlier, MAHATMA has several parameters. Hence, this paper describes experiments performed to find good values for some of these parameters. In these experiments the expansion operator was initially turned off, because this is a computationally expensive operator and we wanted to perform some relatively quick experiments to set other parameters. The expansion operator has been used in later experiments, though. The leaf operators (mutation and crossover operators) were set during experiments with a genetic algorithm that discovers protein motifs in a standalone fashion, without being hybridized with GP as in MAHATMA (Tsunoda and Lopes 2005).

The initial parameter settings are: number of generations, 20; population size, 500; structural crossover and mutation probability, 60% each; hill climbing, 10% probability; leaf crossover and mutation probabilities, 20 and 70%; stochastic tournament size, 3%; edition active and expansion deactivated. From now on these parameter values will be referred to as the initial values.

To evaluate the predictive accuracy of the classification algorithm we used the well-known fivefold cross-validation method (Witten and Frank 2005) in all experiments reported in this paper. The average values of specific predictive accuracy measures on the test set (unseen during training) over all five folds are the so-called cross-validated predictive accuracy measures. Although tenfold cross-validation is more common in the literature, we used fivefold cross-validation to reduce the computational time taken by the experiments (since we have done many experiments with different parameter values).

For each fold of the cross-validation procedure, the GP method is run once for each class, since each run has to use that fold's training set to find motifs (rules/patterns) discriminating one class (the "positive" class) from all the other classes (collectively considered the "negative class"). Hence, each run of the GP method discovers a set of possible rules (each represented by an individual) which tend to occur in proteins of the positive class and tend not to occur in proteins from the other classes.

The number of motifs discovered for each class (i.e., the number of individuals output by MAHATMA as the discovered "solution") is a user-defined parameter. At present this parameter specifies the same value for all the classes, for the sake of simplicity, but in future research it might be interesting to allow the number of motifs discovered for each class to be variable, depending for instance on the number of examples available for each class. The returned motifs (individuals) are the ones with the best fitness in the last generation.

For each cross-validation fold, once a set of motifs has been discovered (from the training set) for each class, the entire set of discovered motifs can be used to classify examples in the test set of the current cross-validation fold

[2] Available at http://www.pdb.org/pdb/home/home.do.

(i.e., examples whose class is unknown by the system) in two different ways, as follows.

Firstly, each individual can be interpreted as a rule of the form IF <motif_condition> THEN <motif_class>, where motif_condition is the logical expression associated with the motif involving all nodes of the corresponding individual, see, e.g., Fig. 1; and motif_class is the positive class associated with the rule that was discovered. Hence, this kind of rule suggests that if the logical expression associated with a rule antecedent is satisfied by an example (protein), then the rule predicts the motif's class for that example.

The entire set of rules corresponding to all discovered motifs (for all classes) is then used as follows. For each new example in the test set, the example is submitted to all discovered rules and the system computes the number of rules in each class whose antecedents are satisfied by the example. Finally, the test example is assigned the class with the highest number of rules satisfied by the example, subject to the condition that at least 60% of the rules for the chosen class have to be satisfied by the example. If this latter condition is not satisfied, then the test protein is simply assigned the most frequent class in the training set (in our dataset, EC3). Note that in this case the motif (individuals/rules) discovered by MAHATMA are directly used for the classification of test examples.

A second and very different approach to classify examples in the test set consist of passing each of the discovered motifs to a separate classification algorithm, which simply uses the set of discovered motifs as a set of predictor attributes. That is, in this approach each motif is used as a binary attribute, taking the value 1 or 0 to indicate whether or not the pattern (motif) occurs in each example (protein). In this case in principle any classification algorithm can be used with the set of motifs discovered by MAHATMA.

4 Computational results

We report first the results of experiments with the first approach, using the discovered motifs to directly classify examples in the test set. Results of experiments with the second approach, using a well-known decision tree induction algorithm as the classification algorithm, are reported later in the paper.

Each result table reports the following cross-validated measures of predictive accuracy: sensitivity (Sn), specificity (Sp), performance (*P*) (Sn multiplied by Sp) (Lopes 1996) and hit rate (HR). Sn and Sp are defined in Eq. 2 and 3. Mahatma's HR is defined as Eq 4, where proteinCount is the count of proteins in test set. In Eqs. 2, 3, 4 the terms TP, FP, TN and FN denote, respectively, the numbers of true positives, false positives, true negatives and false negatives observed in the test set; these are well-known terms used to measure predictive accuracy in the classification task of machine learning and data mining, see, e.g., (Witten and Frank 2005).

We have bold emphasized the best results (better performance) in each of the tables of results. In each of the Tables 3, 4, 5, 6, 7, the best value of the performance measure (*P*) can be considered as statistically relevant, as there is no overlap of the confidence intervals (consisting of ±one standard deviation around the mean computed by the cross-validation procedure) associated with the best *P* value and the other *P* values in the table.

$$\mathrm{Sp} = \frac{\mathrm{TN}}{(\mathrm{TN} + \mathrm{FP})} \tag{2}$$

$$\mathrm{Sn} = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FN})} \tag{3}$$

$$\text{Hit rate} = \frac{\mathrm{TP}}{\mathrm{proteinCount}} \tag{4}$$

The first step was to find a good value for the number of generations (*G*) and population size (PS). The results obtained via fivefold cross-validation are reported in Table 3.

The second step was to adjust structural crossover (SC) and mutation (SM) probabilities (%). The results are reported in Table 4.

The third step adjusted the hill climbing (HC) probability (%). As shown in Table 5, higher values for this parameter do not assure better results. In fact, when we used 70%, performance decreased significantly. This happens because this parameter does not guarantee the offspring's improvement. It simply states that a parent will be copied for next generation if the offspring has lower fitness than that parent.

Since the experiments that generated Table 5 led to the conclusion that higher predictive accuracy was achieved with hill climbing probability 40%, instead of 0 or 70%, this value (40%) was used to run the experiments summarized in Tables 6 and 7.

The fourth step fixed a good value for the parameter tournament size. This parameter was given special attention, because it is potentially one of the most important

Table 3 MAHATMA's performance varying number of generations and population size

G	PS	Sn (%)	Sp (%)	*P* (%)	HR (%)
20	**500**	**87.28 ± 0.12**	**43.35 ± 0.31**	**61.51 ± 0.21**	**79.03 ± 0.74**
40	250	86.85 ± 0.12	37.60 ± 0.30	57.15 ± 0.20	78.28 ± 0.81
50	200	86.87 ± 0.12	42.37 ± 0.36	60.67 ± 0.26	77.40 ± 1.12
70	150	85.56 ± 0.12	32.32 ± 0.30	52.59 ± 0.20	77.30 ± 0.79

Table 4 MAHATMA's performance varying structural mutation and crossover probabilities

SM	SC	Sn (%)	Sp (%)	*P* (%)	HR (%)
30	60	87.17 ± 0.12	38.37 ± 0.32	57.83 ± 0.22	78.68 ± 0.99
20	70	87.70 ± 0.11	41.62 ± 0.30	60.42 ± 0.21	79.85 ± 1.01
10	80	86.70 ± 0.12	36.18 ± 0.32	56.01 ± 0.21	77.76 ± 0.98
60	**60**	**87.28 ± 0.11**	**43.35 ± 0.32**	**61.51 ± 0.23**	**79.03 ± 0.74**
60	30	87.18 ± 0.11	42.09 ± 0.32	60.58 ± 0.23	78.40 ± 0.92
70	20	87.49 ± 0.10	39.61 ± 0.31	58.87 ± 0.22	79.32 ± 0.68
80	10	88.31 ± 0.10	42.34 ± 0.28	61.15 ± 0.20	81.34 ± 0.87

Table 5 MAHATMA's performance varying hill climbing probability

HC (%)	Sn (%)	Sp (%)	*P* (%)	HR (%)
0	87.31 ± 0.11	42.38 ± 0.30	60.83 ± 0.21	79.19 ± 0.89
40	**86.65 ± 0.12**	**47.09 ± 0.34**	**63.88 ± 0.25**	**76.95 ± 0.84**
70	87.12 ± 0.11	38.52 ± 0.34	57.93 ± 0.24	77.98 ± 0.85

Table 6 MAHATMA's performance varying tournament size

TS (%)	Sn (%)	Sp (%)	*P* (%)	HR (%)
1	86.85 ± 0.14	37.60 ± 0.34	57.15 ± 0.27	77.87 ± 0.89
3	**86.65 ± 0.12**	**47.09 ± 0.34**	**63.88 ± 0.25**	**76.95 ± 0.84**
5	87.10 ± 0.14	42.31 ± 0.39	60.71 ± 0.31	77.99 ± 1.08
7	86.94 ± 0.13	42.43 ± 0.37	60.74 ± 0.28	77.05 ± 0.71

Table 7 MAHATMA's performance varying number of motifs per class

NM	Sn (%)	Sp (%)	*P* (%)	HR (%)
1	86.15 ± 0.17	28.76 ± 0.25	49.78 ± 0.15	78.42 ± 0.91
5	87.26 ± 0.16	35.68 ± 0.27	55.80 ± 0.16	81.19 ± 0.87
10	**86.65 ± 0.12**	**47.09 ± 0.34**	**63.88 ± 0.25**	**76.95 ± 0.84**
15	87.64 ± 0.15	41.98 ± 0.30	60.66 ± 0.21	80.25 ± 0.75
20	87.11 ± 0.16	41.61 ± 0.31	60.20 ± 0.22	81.15 ± 0.73

parameters of an evolutionary algorithm. The reason is that this parameter directly determines the selective pressure of the algorithm. The larger the tournament size, the larger the selective pressure. The results of experiments with different values of tournament size are reported in Table 6.

Surprisingly, the value of tournament size had little impact in the predictive accuracy. In any case, we decided to fix the default value of this parameter to 3%, since this value led to slightly higher predictive accuracy.

Having fixed this parameter, the next experiment evaluated the influence of the expansion operator in the classification accuracy. The expansion operator was somewhat effective, leading to a slight increase of the predictive accuracy (performance of 64.69%), but the processing time increased exponentially (22 h instead of 37 min).

Finally, we performed experiments to determine the influence—in the predictive accuracy—of another important parameter of the algorithm, the number of motifs (NM) used for each class. In the experiments reported so far this parameter was set to five motifs per class. The new experiments produced the results shown in Table 7.

As it can be observed in Table 7, there was some variation in predictive accuracy when the number of motifs changed. However, three values of this parameter were considerably more successful than the value of 5 which had been used in earlier experiments. Hence, it is important to return a larger number of motifs per class, in order to give more predictor attributes to the classification algorithm.

For comparison purposes, as described earlier, the list of the top ten best motifs (or rules) for each class, as well as the corresponding training results and test sets of course, were converted as input (arff file) for the WEKA data mining tool, in order to allow the discovered motifs to be used as predictor attributes by a separate classification algorithm. As the classification algorithm we chose J4.8, a Java implementation of the very well-known C4.5 algorithm (Quinlan 1993). This choice was motivated for the following reasons. First, J4.8 produces a decision tree, a classification model that tends to be comprehensible to the user, allowing him/her to interpret discovered knowledge. This is important in bioinformatics applications such as protein function prediction (Freitas et al. 2010), and also in data mining in general, where the goal is to give the user some new insight about the predictive relationships that hold in the data. Second, J4.8 is available in the WEKA data mining tool (Witten and Frank 2005), which has the advantage of being a public domain and widely used tool.

Using the J48 classification algorithm, with WEKA default parameters values and active pruning, the results were sensitivity (Sn), specificity (Sp) and performance (*P*) values of 82.88 ± 6.03, 96.81 ± 1.78 and 89.50 ± 3.02, respectively; and a hit rate of 85.36% ± 2.59. The average size of the trees and the average count of leaf nodes are, respectively, 648 and 324. These results are in general better than the results produced using the discovered motifs directly for classification, as reported in Tables 3, 4, 5, 6, 7, with the exception that using J4.8 led to a somewhat smaller Sn than the best results in Tables 3, 4, 5, 6, 7.

5 Related work

Broadly speaking, MAHATMA is related to three types of algorithms in the literature. First, since MAHATMA addresses the classification task of data mining, it is related to GP algorithms for classification. There are innumerous

GP algorithms for classification, a good recent review of which can be found in (Espejo et al. 2010). However, in general those GP algorithms have not been applied to protein function prediction. The second type of algorithm related to MAHATMA involves classification algorithms that have been applied to protein function prediction, but are not based on GP. A review of such algorithms can be found in (Freitas and de Carvalho 2007).

It should be noted that neither of the above two broad types of algorithms can be directly applied to the protein function prediction problem in the form addressed by MAHATMA. The main reason for this is that classification algorithms for protein function prediction (whether or not they are based on GP) in general require the training examples to be of fixed length, with the same number of (abstract) attributes per example. In contrast, MAHATMA uses training examples of variable length, since examples represent directly the primary sequences of proteins (and different proteins have primary sequences of different lengths), and operates directly on the amino acid sequence, rather than using abstract attributes. Hence, instead of using a fixed number of abstract attributes predefined by the user, MAHATMA automatically constructs motifs that are used as predictor attributes.

The third type of algorithm related to MAHATMA involves algorithms to discover motifs from proteins' primary sequences. According to Leung and Chin (2006) there are two general approaches to extracting sequence motifs in proteins. In the first one, given sequences are compared or aligned to find local similarities; subsequences common to the dataset are detected and checked. In the second approach, candidate patterns in the solution space are ennumerated, and "appropriate" or "better" ones are chosen based on their fitness scores. MAHATMA is different from well-known systems such as MEME because it uses positive and negative examples, in the context of a trainable classification algorithm, in order to discover motifs. Leung and Chin state that MEME, in some cases, cannot guarantee finding a motif in a reasonable time, especially when the motif is too long.

6 Conclusions

We have proposed a Genetic Programming method (which can also be regarded as a hybrid genetic programming/genetic algorithm system) for protein motif discovery, aiming to classify proteins with unknown functional class.

We have performed experiments to adjust the parameters of our method in an enzyme subset of the PDB (Protein Data Bank), containing 8,399 enzymes.

The proposed MAHATMA system uses not only conventional GP (and GA-like) operators, but also operators specifically designed for the problem of finding protein motifs. Despite the complexity of the algorithm, the use of these problem-specific operators was very beneficial in the sense that it allowed MAHATMA to reach better motifs (motifs with higher fitness).

The predictive performance was measured by three different criteria, viz., using sensitivity (Sn), specificity (Sp) and hit rate (HR), in two different scenarios: using the motifs discovered by MAHATMA directly for classification, and using the discovered motifs as predictor attributes for another classification algorithm. Overall the latter approach led to better Sp and much better HR, but worser Sn.

Future work includes more extensive tests of the system in datasets involving enzymes' secondary structures (Kaminska et al. 2009) and comparisons with other methods (e.g., comparing the motifs found by MAHATMA and MEME using the well-known C4.5 classification method). Also, it is intended to apply this system to alternative sets of proteins, like transmembranes, globins, hormones and others.

In addition, the fact that the best predictive accuracy was obtained when the motifs discovered by MAHATMA were used as predictor attributes by another classification algorithm, rather than using the motifs discovered directly for classification, suggests that the procedure currently used to combine the predictions of the set of discovered motifs could be improved. This will also be object of future research.

References

Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. J Mol Biol 215:403–410

Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic programming: an introduction. Morgan Kaufmann, San Mateo, CA

Branden CI, Tooze J (1999) Introduction to protein structure. Garland, New York

Chua H, Sung W, Wong L (2006) Exploiting indirect neighbors and topological weight to predict protein function from protein interactions. Bioinformatics 32(13):1623–1630. doi:10.1093/bioinformatics/btl145

desJardins M, Karp PD, Krummenacker M, Lee TJ (1997) Prediction of enzyme classification from protein sequence without the use of sequence similarity. ISMB-97 Proceedings, pp 92–99

Eiben AE, Smith JE (2003) Introduction to evolutionary computing, 2nd printing. Natural computing series. Springer, Berlin

Espejo PG, Ventura S, Herrera F (2010) A survey on the application of genetic programming to classification. IEEE Trans Syst Man Cybern Part C Appl Rev 40(2):121–144. doi:10.1109/TSMCC.2009.2033566

Freitas AA, de Carvalho ACPLF (2007) A tutorial on hierarchical classification with applications in bioinformatics. In: Taniar D (ed) Research and trends in data mining technologies and applications, Idea Group, pp 175–208

Freitas AA, Wieser DC, Apweiler R (2010) On the importance of comprehensible classification models for protein function prediction. IEEE/ACM Trans Comput Biol Bioinform 7(1):172–182. doi:10.1109/TCBB.2008.47

Friedberg I (2006) Automated protein function prediction—the genomic challenge. Brief Bioinform 7(3):225–242. doi:10.1093/bib/bbl004

Goldberg DE (1989) Genetic algorithms in search optimization and machine learning. Addison-Wesley, Reading

Hsu WH (2009) Genetic programming. In: Wang J (ed) Encyclopedia of data warehousing and mining, 2nd edn. Idea Group Inc. Global, pp 926–931

Izrailev S, Farnum MA (2004) Enzyme classification by ligand binding. Proteins Struct Funct Bioinform 57(4):711–724. doi:10.1002/prot.20277

Jensen LJ, Gupta R, Blom N, Devos D, Tamames J, Kesmir C, Nielsen H, Staerfeldt HH, Rapacki K, Workman C, Andersen CAF, Knudsen S, Krogh A, Valencia A, Brunak S (2002) Prediction of human protein function from post-translational modifications and localization features. J Mol Biol 319:1257–1265. doi:10.1016/S0022-2836(02)00379-0

Kaminska KH, Milanowska K, Bujnicki JM (2009) The basics of protein sequence analysis. In: Bujnicki JM (ed) Prediction of protein structures, functions, and interactions, pp 1–38. doi:10.1002/9780470741894

Koza JR (1992) Genetic programming—on the programming of computers by means of natural selection. MIT Press, Cambridge

Koza JR (1994) Genetic programming ii: automatic discovery of reusable programs. MIT Press, Cambridge

Larose DT (2006) Data mining methods and models. Wiley and Sons, Hoboken, NJ

Lehninger AL, Nelson DL, Cox MM (1998) Principles of biochemistry, 2nd edn. Worth Publishers, New York

Lesk AM (2001) Introduction to protein architecture. Oxford University Press Inc., New York

Leung CM, Chin FYL (2006) Algorithms for challenging motif problems. J Bioinform Comput Biol 4:43–58. doi:10.1142/S0219720006001692

Lopes HS (1996) Analogia e Aprendizado Evolucionário: uma aplicação em diagnóstico clínico. PhD thesis, Brazil (in Portuguese)

Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical report Caltech Concurrent Computation Program, No. 826, CA

Nisbet R, Elder J, Miner G (2009) Statistical analysis and data mining applications. Elsevier, San Diego, CA

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo, CA

RCBS (2010) Research collaboratory for structural bioinformatics (RCSB) website. Available at http://www.pdb.org/pdb/home/home.do

Rost B, Liu J, Nair R, Wrzeszczynski KO, Ofran Y (2003) Automatic prediction of protein function. CMLS Cell Mol Life Sci 60:2637–2650

Santos CT, Bazzan ALC, Lemke N (2009) Automatic classification of enzyme family in protein annotation. Lect Notes Comput Sci 5676:86–96. doi:10.1007/978-3-642-03223-3_8

Silla Jr CN, Freitas AA (2010) A survey of hierarchical classification across different application domains. Data Min Knowl Discov (in press)

Tsunoda DF, Lopes HS (2005) Automatic motif discovery in an enzyme database using a genetic algorithm-based approach. Soft Comput Fusion Found Methodol Appl 10(4):325–330. doi:10.1007/s00500-005-0490-z

Tsunoda DF, Freitas AA, Lopes HS (2009) MAHATMA: a genetic programming-based tool for protein classification. In: Proc 2009 ninth international conference on intelligent systems design and applications (ISDA-09), IEEE Press, pp 1136–1142

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Mateo, CA

Zhao XM, Wang Y, Chen L, Aihara K (2008) Protein function prediction with high-throughput data. Amino Acids 35(3):517–530. doi:10.1007/s00726-008-0077-y