2009 Ninth International Conference on Intelligent Systems Design and Applications

MAHATMA: a Genetic Programming-Based Tool for Protein Classification

Denise F. Tsunoda
Federal University of Parana
dtsunoda@ufpr.br

Alex A. Freitas
University of Kent
A.A.Freitas@kent.ac.uk

Heitor S. Lopes
Federal University of Technology
hslopes@utfpr.edu.br

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 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 (Se) and specificity (Sp). The best results obtained for the enzyme dataset suggest that the proposed evolutionary computation method is very effective to find predictive features (motifs) for protein classification.

1. Introduction

This paper proposes a computational tool based on an evolutionary computation technique, more precisely a genetic programming method, specially devised for the automatic discovery of protein motifs using as input the primary structure of proteins.

Proteins are responsible for several functions 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, also called primary structure, is inextricably linked to its function [1]. 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 [2]. Proteins are grouped into super families, families and subfamilies according to these biological functions [3,4,5].

Despite the existence of several methods to solve the protein function prediction problem [6,7], it still remains one of the main challenges in the current post-genomic era.

The proposed tool – MAHATMA – finds sequences of amino acids (features or motifs) 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.

2. Method

Genetic programming [8, 9] was 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.

2.1. Basic algorithm and individual representation

MAHATMA – Memetic Algorithm-based Highly Adapted Tool for Motif Ascertainment – is a genetic programming (GP) based tool [8, 10]. In GP – like in other types of evolutionary algorithms – each individual corresponds to a candidate solution to the target problem. In this work the goal of the GP method is to find a set of rules combining protein motifs

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.14

which, when used as predictive features, lead to a high protein-classification accuracy. In this work, an individual is represented by a tree (Figure 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.

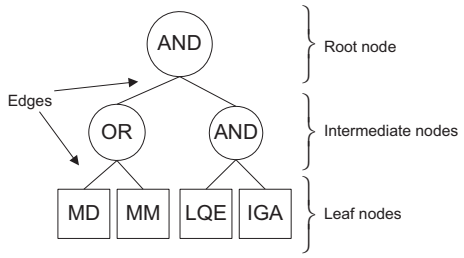

Figure 1. MAHATMA individual representation

Hence, each individual represents the antecedent (IF part) of an IF-THEN classification rule consisting of a motif formed by applying logical operations to amino acid sequences. For instance, the individual shown in Figure 1 can be read as the rule antecedent: IF "(a protein has the aminoacid sequence MD or MM) and (a protein has the aminoacid sequences LQE and IGA)".

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.

2.2. Selection Method and Genetic Operators

The system uses stochastic tournament selection, which works as follows [11]. 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 k, the higher the selective pressure will be, possibly leading the algorithm to stick rapidly in a "local maximum". On the other hand, a k too small will impose no selective pressure, turning the method into a random search.

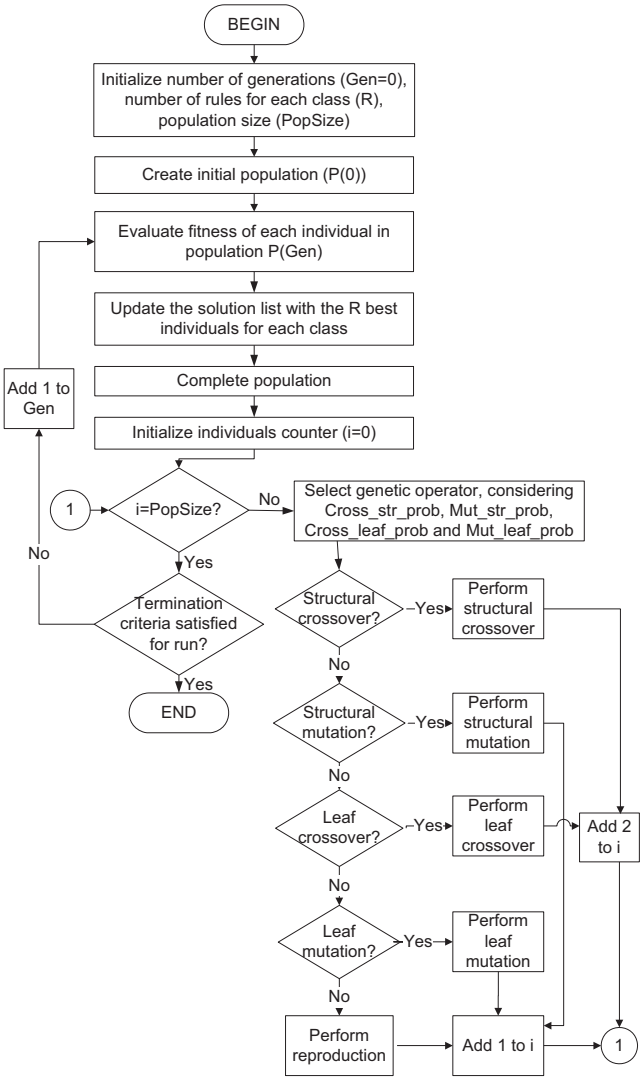

Figure 2. MAHATMA flowchart

We emphasize that MAHATMA has two kinds of operators: structural operators (usual in GP [8]) and leaf operators (based on genetic algorithms [13]). The structural operators are: reproduction, crossover, mutation, editing and encapsulation.

The reproduction operator just copies a selected individual to the next generation. The encapsulation keeps the best M motifs found throughout the evolutionary process, where M is a user-defined parameter. In other words, the encapsulation operator identifies a potentially useful subtree and gives it a tag so that it can be referenced and used later.

The leaf operators modify the sequence of amino acids by genetic operators (e.g. crossover and mutation) in order to produce offspring [12, 13].

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 "current generation" structure in Figure 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 "next generation" structure.

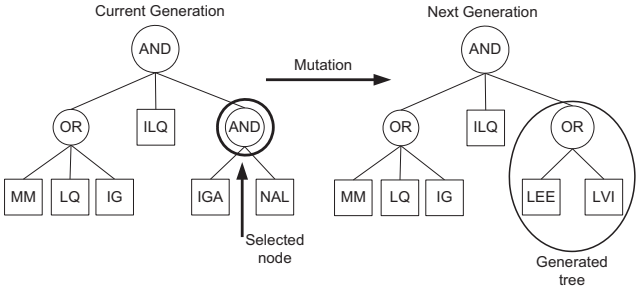

Figura 3. MAHATMA structural mutation operator

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 "current generation" structure in Figure 4, one random point in each parent is select. Each of these points is a rooted subtree crossover point. Figure 4 "next generation" shows the two offspring resulting from crossover.

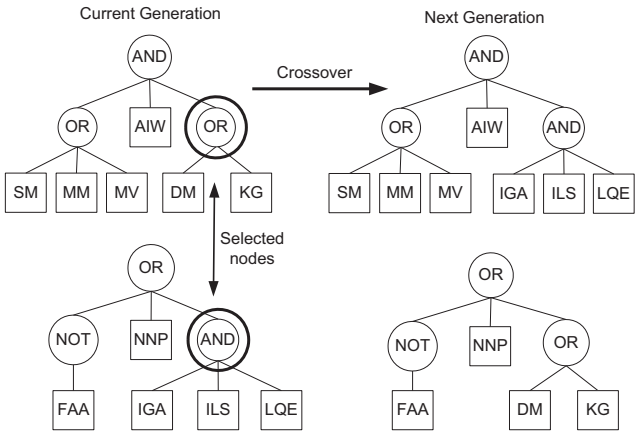

Figure 4. MAHATMA structural crossover operator

Edition is an asexual operator and it recursively applies a set of simplifying operations 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 example of this operator.

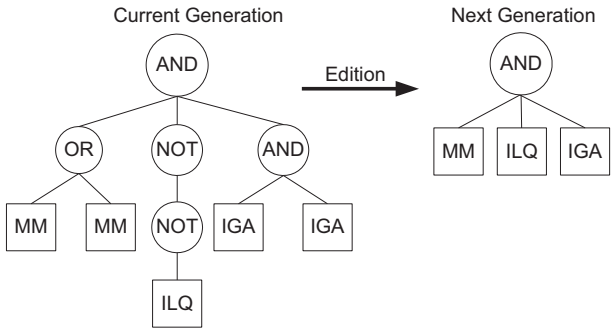

Figure 5. MAHATMA edition operator

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, 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 [10]. 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 Figure 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 Figure 6.

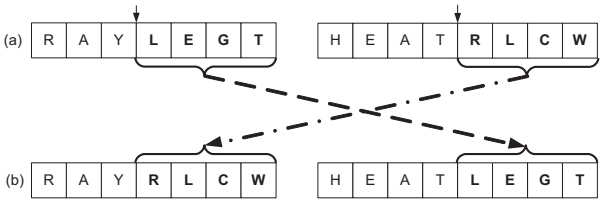

Figure 6. One-point crossover between variable-length parents: (a) original parents, (b) offspring

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 worse 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 worse 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 [14]. 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 individual's fitness is recomputed. If the new fitness is worse 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 worse 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.

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 enzymes 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 i-th class where the motif occurs in the protein's primary sequence. Secondly, the EA 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 the equation 1, where F_i is the relative frequency of the individual's motif in the i-th 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.

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

Equation 1. Fitness function

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 – as usual in conventional evolutionary algorithms. It is necessary to return a set of motifs, in order to perform a comprehensive classification of proteins into known families. In this work, we return the best M motifs found throughout the evolutionary process, where M is a user-defined parameter.

The set of motifs returned is used for classification as follows. Each returned motif is interpreted as a binary attribute. For each protein in the data being mined, the value of a given attribute is true if its motif occurs in its primary sequence, and false otherwise. Hence, each protein can be described by a set of M binary attributes.

Note that the result returned is used to create a new data set, containing data about the same proteins used to evolve the motifs, but representing those proteins at a higher-level of abstraction, with binary attributes corresponding to the presence or absence of motifs, rather than representing the proteins at the very low-level of abstraction associated with their sequence of amino acids. Once this new, higher-level data set has been produced, the next step is to apply a classification algorithm to it, in order to finally produce a classification model that can predict the class of a protein based on the motifs occurring in it.

We used a well-known five-fold cross validation method [15]. The average error rate on the test set (unseen during training) over all five folds is the so-called cross-validated error rate.

3. 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. This is an enzyme code provided by IUBMB (International Union of Biochemistry and Molecular Biology). 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 [16, 17] 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. [18]. 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 non-standard 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: 1,483 proteins in class EC.1; 1,766 in class EC.2; 3,285 in class EC.3; 675 in class EC.4; 381 in class EC.5 and 209 in class EC.6.

4. Computational results

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 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. Each result table reports sensitivity (Se), specificity (Sp), performance (P) (Se multiplied by Sp) [19] and hit rate (HR). We have bolded the best results (better performance).

The first step was to find a good value for generation number (G) and population size (PS). The results obtained via 5-fold cross-validation are reported in Table 1.

Table 1. Generation number and population size

G	PS	Se (%)	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

The second step was to adjust structural crossover (SC) and mutation (SM) probabilities (%). The results are reported in Table 2.

Table 2. Structural mutation and crossover probabilities

SM	SC	Se (%)	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

The third step adjusted the hill climbing (HC) probability (%). As shown in Table 3, 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 the table 3 leaded to the conclusion that higher classification accuracy was achieved with hill climbing probability 40% - instead of 10%, this value (40%) was used to run the experiments summarized in tables 4 and 5.

Table 3. Hill climbing probability

HC	Se (%)	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

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 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. We have performed experiments with four different values of tournament size, namely 1%, 3%, 5% and 7% of the population size. The results are reported in Table 4.

Table 4. Experiments to adjust the tournament size

TS	Se (%)	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

Surprisingly, the value of tournament size had little impact in the classification accuracy. In any case, we decided to fix the default value of this parameter to 3%, since this value led to slightly higher classification 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 classification accuracy (performance of 64.69%), but the processing time increased exponentially (twenty two hours instead of thirty seven minutes).

Finally, we performed experiments to determine the influence – in the classification accuracy – of another important parameter of the algorithm, the number of motifs (NM) (or rules) used for each class. In the experiments reported so far this parameter was set to 5 motifs per class. The new experiments evaluated four different values of this parameter, namely 1, 5, 10, 15 and 20, which produced the results shown in Table 5.

Table 5. Effect of number of motifs per class

NM	Se (%)	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

As it can be observed in Table 5, there was some variation in predictive accuracy when the number of motifs (rules) 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.

5. Results and discussion

We have proposed a system based on a modified Genetic Programming method for motif discovery, aiming to classify unknown-class proteins.

We have performed experiments to adjust the parameters of our method in an enzyme subset of the PDB, containing 8,399 enzymes, distributed across the six classes as follows: 1,483 proteins in class EC.1; 1,766 in class EC.2; 3,285 in class EC.3; 675 in class EC.4; 381 in class EC.5 and 209 in class EC.6.

The proposed MAHATMA system uses not only conventional GP 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 using sensitivity (*Se*) and specificity (*Sp*) and best results are 86.65±0.12 and 47.09±0.34, respectively.

Future work includes more extensive tests of the system in datasets involving enzymes' secondary structures [20] and comparisons with other methods. Also, it is intended to apply this system to alternative sets of proteins, like transmembranes, globins, hormones and others.

6. References

[1] Lehninger A.L., Nelson D.L. and Cox M.M., *Principles of Biochemistry*. 2nd ed. Worth Publishers, New York, 1998.

[2] Branden, C.I., Tooze, J. *Introduction to protein structure*. Garland Publishing Inc, New York, 1999.

[3] I. Friedberg., "Automated protein function prediction – the genomic challenge". *Briefings in Bioinformatics*, vol. 7, no. 3, 2006, pp. 225-242.

[4] B. Rost, J. Liu, R. Nair, K.O., Wrzeszczynski and Y. Ofran, "Automatic prediction of protein function". *CMLS Cellular and Molecular Life Sciences*, n. 60, 2003, pp. 2637-2650.

[5] L.J., Jensen, R. Gupta, N. Blom, D. Devos, J. Tamames, C. Kesmir, H. Nielsen, H.H. Staerfeldt, K. Rapacki, C. Workman, C.A.F. Andersen, S. Knudsen, A. Krogh, A. Valencia and S. Brunak, "Prediction of human protein function from post-translational modifications and localization features". *J. Mol. Biol*., 319, 2002, pp. 1257-1265.

[6] H. Chua, W. Sung, and L. Wong, "Exploiting indirect neighbors and topological weight to predict protein function from protein interactions". *Bioinformatics*, v. 32, n. 13, 2006, pp. 1623-1630.

[7] X.-M. Zhao, Y. Wang, L. Chen, and K. Aihara, "Protein function prediction with high-throughput data". *Amino Acids*, v. 35, n. 3, 2008, pp. 517-530.

[8] Koza, J.R. *Genetic Programming – on the programming of computers by means of natural selection*, The MIT Press, Cambridge, 1992.

[9] Koza, J.R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. The MIT Press, Cambridge, 1994.

[10] W.H. Hsu, "Genetic Programming". *Encyclopedia of Data Warehousing and Mining*. In: Wang, J. (Ed.), 2nd ed. Idea Group Inc. Global, 2009, pp. 926-931.

[11] Banzhaf, W., P. Nordin, R.E. Keller, and F.D. Francone, *Genetic Programming: an Introduction*, Morgan Kaufmann, San Mateo, 1998.

[12] Goldberg, D.E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, Reading, 1989.

[13] Larose D.T., *Data Mining Methods and Models*, John Wiley & Sons, Hoboken, New Jersey, 2006.

[14] Moscato, P. *On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms*. Technical Report Caltech Concurrent Computation Program, n. 826, California, 1989.

[15] Witten I.H., Frank E., *Data mining: practical machine learning tools and techniques*, 2nd ed., Elsevier, Morgan Kaufmann, USA, 2005.

[16] A.A. Freitas and A.C.P.L.F. de Carvalho, "A Tutorial on Hierarchical Classification with Applications in Bioinformatics". In: D. Taniar (Ed.) *Research and Trends in Data Mining Technologies and Applications*, Idea Group, 2007, pp. 175-208.

[17] N. Holden and A.A. Freitas, "Improving the Performance of Hierarchical Classification with Swarm Intelligence". In: E. Marchiori and J.H. Moore (Eds.) Proc. Sixth European Conf. on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio-2008), *Lecture Notes in Computer Science*, n. 4973, 2008, pp. 48-60.

[18] W. Weinert and H.S. Lopes, "Neural networks for protein classification". *Applied Bioinformatics*, v. 3, n.1, 2004, pp. 38-41.

[19] Lopes, H.S. *Analogia e Aprendizado Evolucionário: uma Aplicação em Diagnóstico Clínico*. PhD Thesis, Brazil, 1996.

[20] K.H. Kaminska, K. Milanowska and J.M. Bujnicki, The Basics of Protein Sequence Analysis. In: J.M. Bujnicki (Ed.) *Prediction of Protein Structures, Functions, and Interactions*, 2009, pp. 1-38.