PROTEAS: A Finite State Automata based data mining algorithm for rule extraction in protein classification

Fotis E. Psomopoulos

1st author's affiliation
Dept. Electrical and Computer Engineering
Aristotle University of Thessaloniki
+30 2310 99 6362

fpsom@auth.gr

Pericles A. Mitkas

2nd author's affiliation
1st line of address
2nd line of address
+30 2310 99 6390

mitkas@eng.auth.gr

ABSTRACT

An important challenge in modern functional proteomics is the prediction of the functional behavior of proteins. Motifs in protein chains can make such a prediction possible. The correlation between protein properties and their motifs is not always obvious, since more than one motifs may exist within a protein chain. Thus, the behavior of a protein is a function of many motifs, where some overpower others. In this paper a data mining approach for a motif-based classification of proteins is presented. A new classification algorithm that induces rules and exploits finite state automata is introduced. First, data are modeled by terms of prefix tree acceptors, which are later merged into finite state automata. Finally, a new algorithm is proposed, for the induction of protein classification rules from finite state automata. The data mining model is trained and tested using various protein and protein class subsets, as well as the whole dataset of known proteins and protein classes. Results indicate the efficiency of our technique compared to other known data mining algorithms.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – *Data mining, Scientific databases.*

General Terms

Algorithms, Languages

Keywords

Bioinformatics databases, Finite State Automata, Mining methods and algorithms, Classification rules.

1. INTRODUCTION

The greatest achievement in bioinformatics is, without any doubt, the decoding of the DNA. Knowledge of the genetic code enables a better understanding of the mechanism for the creation of all proteins necessary for a cell. However, knowing how a protein is created does not provide any information on how it will be used. All cell functions are performed by specifically designed proteins and not by DNA or RNA. The genetic code does not specify how, where and which proteins will react; it does not even specify if they will ever be created in the first place. For these reasons, one of the greatest challenges in modern bioinformatics is the exact and reliable modeling of protein behavior and functionality.

Proteins are large molecules composed of one or more chains of amino acids in a specific order. The order is determined by the base sequence of nucleotides in the gene that codes the protein. Figure 1 displays the 499 amino acid sequence that forms the P04591 protein chain. Each amino acid in the sequence is represented by its formal abbreviation.

```
          10         20         30         40         50         60
           |          |          |          |          |          |
GARASVLSGG ELDRWEKIRL RPGGKKKYKL KHIVWASREL ERFAVNPGLL ETSEGCRQIL

          70         80         90        100        110        120
           |          |          |          |          |          |
GQLQPSLQTG SEELRSLYNT VATLYCVHQR IEIKDTKEAL DKIEEEQNKS KKKAQQAAAD

         130        140        150        160        170        180
           |          |          |          |          |          |
TGHSNQVSQN YPIVQNIQGQ MVHQAISPRT LNAWVKVVEE KAFSPEVIPM FSALSEGATP

         190        200        210        220        230        240
           |          |          |          |          |          |
QDLNTMLNTV GGHQAAMQML KETINEEAAE WDRVHPVHAG PIAPGQMREP RGSDIAGTTS

         250        260        270        280        290        300
           |          |          |          |          |          |
TLQEQIGWMT NNPPIPVGEI YKRWIILGLN KIVRMYSPTS ILDIRQGPKE PFRDYVDRFY

         310        320        330        340        350        360
           |          |          |          |          |          |
KTLRAEQASQ EVKNWMTETL LVQNANPDCK TILKALGPAA TLEEMMTACQ GVGGPGHKAR

         370        380        390        400        410        420
           |          |          |          |          |          |
VLAEAMSQVT NSATIMMQRG NFRNQRKIVK CFNCGKEGHT ARNCRAPRKK GCWKCGKEGH

         430        440        450        460        470        480
           |          |          |          |          |          |
QMKDCTERQA NFLGKIWPSY KGRPGNFLQS RPEPTAPPEE SFRSGVETTT PPQKQEPIDK

         490
           |
ELYPLTSLRS LFGNDPSSQ
```

Figure 1. P04591 (Gag polyprotein – GAG_HV1H2) protein chain

Depending on their functionality, proteins are classified into classes (or protein families). All proteins in the same class share strong structure similarities and therefore exhibit similar behavior. Until recently, the biological properties of proteins had to be experimentally determined by means of costly and time-consuming methods. Bioinformatics on the other hand uses computational methods to address such problems.

Prediction of protein functionality is possible because of the occurrence of motifs in the protein chain. Motifs are short amino

acid sequences with specific order. As they are the main factors determining protein behavior, they can be used to predict its main functions. It must be noted that there is not always a direct association between motifs and properties because there may be more than one motif present in a protein chain. As a result, the final properties of the protein are a function of many motifs, with some overpowering or amplifying others.

Motifs can be divided into two main categories: patterns and profiles. Patterns are the simplest form of motifs. An example of a pattern is the following:

```
[IV] - G - x - G - T - [LIVMF] - x(2) - [GS]
```

The expression means that the amino acid in the first position can be either an Isoleucine (I) or a Valine (V), followed only by a Glycine (G). Symbol x is a wild card character and x(2) means that two positions can be occupied by any of the 20 amino acids. On the other hand, profiles are multiple sequence alignments, represented by alignment matrices. The main difference between patterns and profiles is that the latter return a similarity score as opposed to the binary TRUE/FALSE returned by a pattern.

There are two main methods of identifying and discovering motifs in protein sequences, either by unsupervised pattern inferencing, using any known machine learning methods, or by supervised pattern discovery, usually after experiments and expert confirmation. For the second case, the best solution for motifs search is the PROSITE ([11], [30]) database, which contains a comprehensive list of documented protein domains constructed by expert molecular biologists. Similar databases are PFAM ([3], [31]), PRINTS ([1], [32]) and SMART [21].

The most popular methods in protein classification, approach the problem using sequence alignments, either by performing pairwise or multiple alignments (Hidden Markov Models), utilizing mainly heuristic algorithms. However, the use of alignments implies that homologous segments in different protein sequences conserve their order, thus contradicting with the evolutionary genetic recombination and re-shuffling [26]. On the other hand, it is widely accepted that classifiers utilizing alignment-based features (such as motifs) at the low end of running time complexity, are inherently ineffective due to unreliable alignments at low sequence identity [8]. Consequently, computational methods that do not utilize alignments have become the focus in computational biology. These methods originate from Chaos Theory, Kolmogorov complexity [26] and Machine Learning techniques [2], such as text mining [7] and data mining ([12], [16], [25]). So far there are many data mining algorithms for protein classification using motifs in the protein chains, both in artificial intelligence and pattern recognition literature. Some generate decision trees ([27], [24]) others use neural networks [5], statistical models ([10], [18]) or combinations of classification algorithms [9]. However, the majority of the approaches to this problem utilize the actual aminoacid sequence of the proteins in question, in order to extract the appropriate classification models ([4], [17]).

A novel approach to this problem utilizing Finite State Automata to process the motifs that appear in a protein sequence, instead of the actual aminoacids, namely the PROTEAS methodology, is presented in this paper. Initially, protein data from an expert-based database is divided into two disjoint sets, the training and the test set. The training set is used to construct a Prefix Tree Acceptor, which is then transformed into a more generalized Finite State Automaton. Finally, rules extracted from the Automaton are applied on the test set, in order to evaluate the efficiency of the methodology. In this paper, after a brief outline of the PROTEAS methodology, the theoretical concepts concerning Finite State Automata are discussed. Following a detailed description of each step of the process, comparative experimental results are presented together with some conclusions.

2. METHODOLOGY OUTLINE

The main goal of the PROTEAS methodology is the extraction of protein classification rules from a Finite State Automaton, which is a generalized model of the protein data. The methodology is illustrated in a block diagram form in Figure 2. The different steps of the process are described below.

Using a supervised motif database (in our case PROSITE), the train and test datasets are created, which are mutually exclusive. In both cases, the class in which the protein belongs to, as well as the motifs it contains, is known.

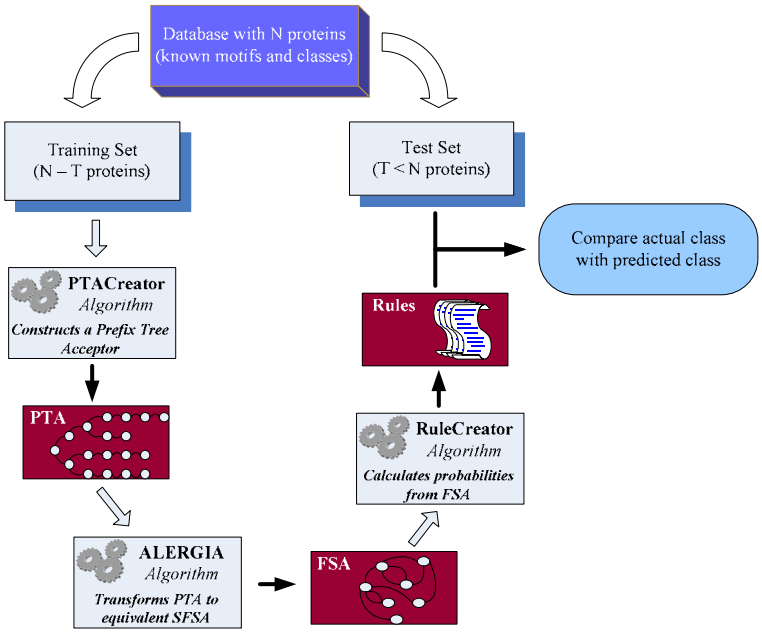

Figure 2. The PROTEAS methodology

In the next step a Prefix Tree Acceptor (PTA) is formed from the protein chains of the train set. The PTA is consequently transformed into an equivalent stochastic deterministic finite state automaton (FSA) using the Alergia algorithm [6]. The equivalence of the two forms allows us to extract information about the proteins directly from the structure of the FSA. As the FSA is a more general representation of the actual dataset used, rather than a strict image like the PTA, it is possible to calculate certain probabilities, such as the probability that a protein chain contains a specific motif, or the probability that a protein chain containing two certain motifs belongs to a specific class.

Using those probabilities we can construct rules that describe the classification of the proteins in the dataset. The rules can be

simple, such as "*If motif a is present in the protein chain, then the protein belongs to class Y*", or more complex, in the form of "*If motifs a and b are present in the protein chain, then the protein belongs in class P with probability p or in class Q with probability q*".

3. FINITE STATE AUTOMATA

At this point a short introduction to the concepts borrowed from the Finite State Automata theory is necessary [15]. A Finite State Automaton is a model of computation consisting of a set of states, a start state, an input alphabet and a transition function that maps input symbols and current states to a next state. Computation begins in the start state with an input string and it changes to new states depending on the transition function. There are many variants, for instance machines giving actions (outputs) associated with transitions (Mealy machine) or states (Moore machine), multiple start states, transitions conditioned on no input symbols (nulls) etc.

The variant used in the proposed methodology is a deterministic finite state automaton (DFA), which has at most one transition for each symbol and state. A DFA M is defined as a 5-tuple of the form:

$$M = (K, \Sigma, \delta, s, F), \text{ where:} \quad (1)$$

K is the set of states,

Σ is the input alphabet,

$s \in K$ is the start state,

$F \subseteq K$ is the set of end states and

δ is the transition function from $K \times \Sigma$ to K.

An input string (composed of symbols contained in Σ) is accepted by the DFA when, while reading it, M goes over (according to δ: $K \times \Sigma \rightarrow K$) from state to state (starting from state s) and ends (exhausting the input string) to any state that belongs to the set of end states F. The most important aspect of the DFA, with regards to other finite state variants, is that from a state q and using the same symbol, the next state is always a specific state q'. In other variants, such as the non-deterministic finite state automata, the transition from a state q using the same symbol, may lead to more than one next states q'.

Finally, another form of DFA is the stochastic deterministic finite state automaton (S-DFA). The main difference between a DFA and an S-DFA is that the latter introduces the concept of the transition probability, i.e. the probability of transition between states according to δ: $K \times \Sigma \rightarrow K$. The result of this probability is that some transitions are favored over others.

4. PREFIX TREE ACCEPTOR

The next step in PROTEAS is the construction of a prefix tree acceptor from a set of protein sequences.

Given a random sequence of symbols (e.g. $w = a_1a_2a_3\ldots a_n$), a prefix of the sequence w is any subsequence wp of w for which:

$$w = w_p\, w' \quad (2)$$

where w' a sequence of any length (i.e. containing any number of symbols, including 0).

It is obvious that for a given set of sequences, it is possible to construct a corresponding set of prefixes. In the new set, every element – prefix will be a sequence of less or at most equal length to the sequence it came from. At the same time, however, the new set allows for the identification of any sequence, using fewer symbols.

The prefixes can be used for the construction of a DFA, which accepts only them, known as a Prefix Tree Acceptor (PTA). As every sequence is a prefix of itself (w′ is null), a PTA can be created from the original sequences. For example, using the following sequences:

$$w_1 = aaaaab$$

$$w_2 = abbaaa$$

$$w_3 = bababa$$

the corresponding PTA is depicted in Figure 3.

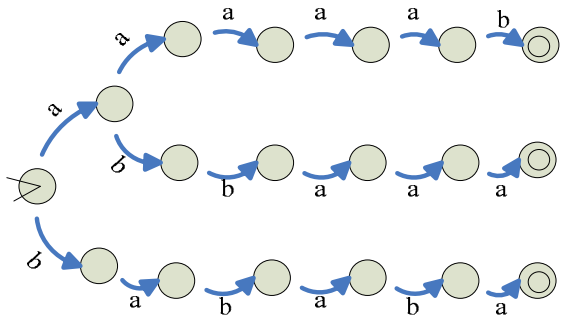

Figure 3. A PTA constructed from the original sequences

However, it is obvious that this PTA can be reduced in size if the corresponding set of prefixes is used for its creation. The set of prefixes with minimum length of the previous sequences, while still having the capability of differentiation, is the following:

$$wp_1 = aa$$

$$wp_2 = ab$$

$$wp_3 = b$$

and the new PTA is shown in Figure 4.

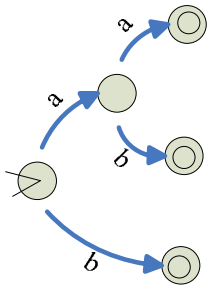

Figure 4. A PTA constructed from the minimum prefixes

In our case, the set of sequences is the set of protein sequences. However, the actual symbols of the amino acids are not used, as every protein sequence is replaced by the corresponding sequence of motifs present in it. The PTA is constructed in the same way using symbol sequences, but in this case each symbol is a motif. Using as an example the protein sequence in Figure 1, the whole amino acid sequence can be replaced by motif PS50158 (amino acids 389 – 427) due to the fact that it is the only motif present.

The PTA construction algorithm is presented in Figure 5. As input, the algorithm requires a set of proteins. Starting from the first protein in the set, the algorithm creates the corresponding sequence of motifs. Then, starting at the start state of the PTA, it examines if there exists a transition to a next state using the first motif in the motif sequence. In case there is such a transition, the algorithm moves to the new state and to the second motif in the sequence. If no such transition exists, the algorithm creates a new state and connects it to the start state using a transition with the first motif. This procedure is repeated until all motif sequences have been traversed, at which point a full PTA has been constructed.

```
algorithm PTACreator
input:
      a set of proteins
output:
      a prefix tree acceptor
begin
  do (for every protein in the training set)
    motifs = findProteinChain(protein)
    currentState = firstNode(PTA)
    do (for every motif m in the sequence motifs)
      if (currentState has no transition for motif m)
        add a new state
        add a transition between the current state and
                                the new state with motif m.
        currentState = newState
      else
        find the end state to which current state
                                    transits using motif m.
        currentState = endState
      end if
    end do
  end do
end algorithm PTACreator
```

Figure 5. `PTACreator` algorithm

Given that there are P proteins in the set, every protein has at most M motifs and there are A different motifs altogether, the time complexity of the PTACreator algorithm is:

$$\boldsymbol{O}(P, M, A) = P \text{ x } M \text{ x } (A - 1) \qquad (3)$$

However, as both M and A can be considered constant, the time complexity of the algorithm rises linearly to the number P of proteins in the set.

5. MERGING TO FSA

The next step in PROTEAS is the construction of an S-DFA, using the previously created PTA. This construction is, in essence, the transformation of the PTA by means of a systematic merging of statistically equivalent states. The algorithm that performs this transformation is the `Alergia` algorithm, proposed by R.C. Carrasco and J. Oncina [6].

However, in order to apply the `Alergia` algorithm, certain statistical parameters must be included in the PTA. These are:

- For every state i, the number of sequences n_i that have arrived there must be known.
- For every state i, the number of sequences $f_i(\#)$ that have ended there must be known.
- For every transition α from any given state i, the number of sequences $f_i(\alpha)$ that have used it must be known.

All these parameters can be easily included during the PTA construction. The ratios $f_i(\alpha)/n_i$ and $f_i(\#)/n_i$ correspond to the probability $p_i(\alpha)$ of the transition α and the ending probability p_{if}.

The `Alergia` algorithm compares pairs of states (q_i, q_j) with $1 \le i \le j - 1$ and $2 \le j \le |\text{PTA}|$, where $|\text{PTA}|$ the number of states in the PTA. Two states are equivalent when they have the same transition probabilities for every symbol α and their next states are also equivalent:

$$q_i \equiv q_j \Rightarrow \begin{cases} p_i(\alpha) = p_j(\alpha) \\ \delta_\alpha(i) \equiv \delta_\alpha(j) \end{cases}, \forall \alpha \qquad (4)$$

However, a strict association (such as equivalence) will lead to an FSA that is in complete correspondence to the original PTA and not a generalization. For this reason, a confidence parameter is introduced, together with the concept of compatibility between states. In this case, two states are merged when they are equivalent within some tolerance limits. The `Alergia` algorithm is detailed in the original paper [6] but for clarity, a brief description is included here.

```
algorithm Alergia
input:
       A: a stochastic PTA
       α: confidence
output:
       stochastic FSA
begin
  do (for j = successor( first node(A)) to lastnode(A))
    do (for i = firstnode(A) to j)
      if (compatible(i, j)
        merge(A, i, j)
        determinize(A)
        exit (i-loop)
      end if
    end for
  end for
  return A
end algorithm Alergia
```

Figure 6. `Alergia` algorithm

The algorithm (Figure 6) accepts as input a PTA. The start state and the next state are checked for compatibility (`Compatible` algorithm). If they are compatible, they are merged, and the new construct is checked for conservation of determinism. Then, the start state and the second state are checked for compatibility, and

so forth, until all states are checked. The same procedure is repeated for every state after the start state including compatibility checks to all remaining states.

The `Compatible` algorithm (Figure 7) checks for the compatibility between two states that are given as input. Initially it checks for the termination probabilities (`Different` algorithm). If the probabilities are different, then the states are not compatible and the algorithm terminates. In the other case, all possible transitions from the original states are checked recursively for compatibility.

Finally, the `Different` algorithm (Figure 8) is simply a Hoeffding limit check [6]. The algorithm receives the statistical parameters of the two states in question as input, and returns a boolean output.

If the number of states in the final FSA is *m*, and the number of different symbols is *A*, then according to algorithm `Compatible`, in each state $(A - 1)$ checks are performed. However, the nodes are checked diagonally ($i < j$). Thus, the time complexity of the `Alergia` algorithm is $\frac{1}{2} \times m \times (m-1) \times (A+1)$. Theoretically, the worst case scenario is a third degree polynomial time complexity. Experimentally however [6], time complexity proves to be linear to the size *m* of the FSA.

```
algorithm Compatible
input:
       i, j: states
       n_i , n_j :number of arriving sequences
       f_i(#), f_j(#):number of ending sequences
       f_i(α), f_j(α): number of sequences using transition
                                          with motif α.
output:
       boolean
begin
  if different (n_i , f_i(#), n_j , f_j(#) )
    return false
  end if
  do (for every α)
    if different (n_i , f_i(α), n_j , f_j(α) )
      return false
    end if
    if not compatible ( δ(i, α), δ(j, α) )
      return false
    end if
  end do
  return true
end algorithm Compatible
```

Figure 7. `Compatible` algorithm

6. EXTRACTION OF SET PROBABILITIES

Following the construction of the FSA, useful information can be extracted from it, such as the probability of a specific set of symbols (motifs) appearing in a random sequence generated by the FSA. In literature, there exist algorithms which compute the probability of a subsequence appearing in a random sequence [14].

```
algorithm Different
input:
       n, n': number of sequences arriving at state
       f, f': number of sequences that
                          end or transit through state
output:
       Boolean
begin
  return | f/n - f'/n' | > sqrt( 1/2 · log 2/α ) · ( 1/sqrt(n) + 1/sqrt(n') )
end algorithm Different
```

Figure 8. `Different` algorithm

However, the input protein data contains no information regarding the order of appearance, in the sense that there is no knowledge of which motif appears first or comes after. Such being the case, it is imperative to transform the existing algorithm to allow both for sequences and sets.

The algorithm proposed by P. Hingston [14] calculates the probability of appearance of an ordered set of symbols in any given sequence. For example, the probability of sequences that contain symbol *x*, followed by symbol *y* is computed and generalized for n symbols (i.e. sequences of the form $x_1, x_2, \ldots x_n, y$).

This method can be generalized to allow for sets of elements, rather than sequences. Given two symbols, *a* and *b*, the probability of them appearing in a sequence, in any order, can be calculated as follows:

$$
\begin{aligned}
P(a+b) = \quad & + P(a \to b) && + && P(b \to a) \\
& - P(a \to b \to a) && - && P(b \to a \to b) \\
& + P(a \to b \to a \to b) && + && P(b \to a \to b \to a) \\
& - \ldots && - && \ldots
\end{aligned}
$$

It is obvious that the probability $P(a + b)$ can be calculated as an infinite sum. Every term adds or subtracts a small quantity, which is due to the overlapping of the probabilities. For example, the sequence *aba* will be taken into account both in the evaluation of $P(a \to b)$ and in the evaluation of $P(b \to a)$, and thus must be subtracted once as the probability $P(a \to b \to a)$. This logic can be generalized for *n* symbols, as shown in the algorithm of Figure 9.

If the final FSA has *K* states and *A* different symbols (motifs), the time complexity to evaluate a probability with *n* symbols is calculated as follows:

```
algorithm ProbExtractor
input:
      a S-DFA
      a set of n motifs
output:
      the probability P(m1+m2+…+mn)
begin
  1st step: Calculate all possible permutations of the n motifs (n!):
    Perm(j), for j = 0, 1, …, (n! – 1)
  2nd step: For every permutation calculate its prefix of length (n-1) and the suffix of length 1:
    Prefix(j), Suffix(j), j = 0, 1, …, (n! – 1)
  3rd step: Initialization:
    i = 0:
      calculate initial probability: P = Σ((-1) · p[Perm(j)])
    i = 1:
      create new sequences based on the original sequences and their prefixes as follows:
                Perm(j) = Perm(j) + Suffix(j), j = 0, 1, …, (n! – 1)
      calculate new probability: P' = Σ((-1) · p[Perm(j)])'
  4th step:
    while (|P – P´| > e)
      i = i + 1
      P = P´
      if ( i mod 2 == 0)
        Perm(j) = Perm(j) + Prefix(j), j = 0, 1, …, (n! – 1)
      else
        Perm(j) = Perm(j) + Suffix(j), j = 0, 1, …, (n! – 1)
      end if
    end while
end algorithm ProbExtractor
```

Figure 9. `ProbExtractor` algorithm

For n symbols, $\sum_{i=1}^{n} \text{comb}\binom{A}{i}$ combinations of symbols must be performed. For each one of the combinations, all permutations are calculated ($i!$ for every combination of i symbols), and for each permutation, i matrix inversions are performed (one for each symbol). Given that the best complexity for matrix inversion is proportional to the cube of the matrix dimension, the total time complexity of the `ProbExtractor` algorithm is:

$$O(A, K, n) = \sum_{i=1}^{n} \left(\frac{A!}{(A-i)!} \cdot i \cdot K^3 \right) \quad (5)$$

7. CLASSIFICATION RULES INDUCTION

After the successful modeling of the protein sequences, first as a PTA and then as an S-DFA, the final step in the PROTEAS methodology is the rule extraction. Rules are logical relations that represent associations between elements in a set ([28], [29]). Usually they are of the form $X \rightarrow Y$, which can be read as "*Elements that satisfy X will possibly satisfy Y also*". Depending on what X and Y actually are, rules can be divided into association rules and classification rules. If X and Y are simple attributes of the elements, then they are association rules. If X is an attribute and Y is a class (or family) of elements, then they are classification rules.

Every rule has two qualitative attributes: *support* and *confidence*. They are defined as:

$$support(X \rightarrow Y) = p(X \rightarrow Y) \quad (6)$$

$$confidence(X \rightarrow Y) = \frac{|X \cup Y|}{|X|}$$

or

$$confidence(X \rightarrow Y) = \frac{support(X \rightarrow Y)}{support(X)} \quad (7)$$

In practice, *support* signifies the frequency of the appearance of the rule, where *confidence* represents the weight of the rule.

Finally, another attribute of the rules, used for their evaluation, is *interest*, defined as:

```
algorithm RuleCreator
input:
       m: maximum number of motifs in each rule
       c: maximum number of classes in each rule
       mininterest: interest threshold
output:
       set of rules
begin
  1st step:
    For every combination Y of the families (from per 1 to per c), calculate support(Y)
  2nd step:
    Create total FSA (for all families involved) and for every combination X of motifs (from per 1 to per m),
                                                                          calculate support(X).
  3rd step:
    For every combination Y of the families (from per 1 to per c) create a local FSA.
          From this FSA and for every combination X of motifs (from per 1 to per m),
          calculate the attributes of rule X  Y. If interest (X  Y) is less than mininterest, discard the rule.
end algorithm RuleCreator
```

Figure 10. `RuleCreator` algorithm

$$interest(X \rightarrow Y) = \frac{p(X \cup Y)}{p(X) \cdot p(Y)} = \frac{support(X \rightarrow Y)}{support(X) \cdot support(Y)} \quad (8)$$

Interest represents the dependency between X and Y. The closer interest is to 1, the more independent are X and Y. By using a threshold in *interest*, it is possible to extract only rules that are applied to independent attributes, i.e. strong and interesting rules:

$$\text{If } \left| \frac{p(X \cup Y)}{p(X) \cdot p(Y)} - 1 \right| \geq min\_interest$$

$$\text{then rule } X \rightarrow Y \text{ is interesting.} \quad (9)$$

In the case of protein data, X is a combination of motifs, and Y is a combination of protein classes. It is obvious that the rules created in this way are classification rules and their attributes can be defined as follows:

$$confidence(X \rightarrow Y) = \frac{\#\ proteins\ that\ contain\ motif\ X\ AND\ belong\ to\ class\ Y}{\#\ proteins\ that\ contain\ X} \quad (10)$$

$$support(X \rightarrow Y) = \frac{\#\ proteins\ that\ contain\ motif\ X\ OR\ belong\ to\ class\ Y}{\#\ proteins\ in\ the\ set} \quad (11)$$

$$support(X) = \frac{\#\ proteins\ that\ contain\ motif\ X}{\#\ proteins\ in\ the\ set} \quad (12)$$

$$support(Y) = \frac{\#\ protein\ that\ belong\ to\ class\ Y}{\#\ proteins\ in\ the\ set} \quad (13)$$

It must be noted that instead of a single motif or class, X and Y can represent a set of motifs or classes, respectively.

These attributes can be evaluated from the set probabilities calculated from the FSA, which can be redefined using the protein data.

$$p(X, Y) = \frac{\#\ proteins\ that\ contain\ motif\ X\ AND\ belong\ to\ class\ Y}{\#\ proteins\ that\ belong\ to\ class\ Y} \quad (14)$$

In this case, if an S-DFA is constructed using only three protein families (namely Y_1, Y_2 and Y_3), then from this automaton, the following probabilities can be calculated:

$$p(X, Y_1 \cup Y_2 \cup Y_3) = \frac{\#\ proteins\ that\ contain\ X\ AND\ belong\ to\ Y_1 \cup Y_2 \cup Y_3}{\#proteins\ that\ belong\ to\ Y_1 \cup Y_2 \cup Y_3} \quad (15)$$

for every motif, or combination of motifs, X which appears in the input sequences. It is obvious that if in the FSA construction all available protein families are used, then the probability will be transformed to:

$$p(X) = \frac{\#\ proteins\ that\ contain\ motif\ X}{\#\ proteins\ in\ the\ set} \quad (16)$$

In this way, it is established that the attribute *support*(X) can be directly calculated from the "total" FSA. On the other hand, the attribute *support*(Y) can be evaluated during the PTA creation, since it is merely a recording of how often a protein family is encountered:

support(Y) =

$$p(Y) = \frac{\#\ proteins\ that\ belong\ to\ class\ Y}{\#\ proteins\ in\ the\ set} \quad (17)$$

The *confidence* attribute can be evaluated from a "local" FSA, in the sense that it is constructed using a subset of the protein families:

$$confidence(X \rightarrow Y) = \frac{support(X, local\ FSA) \cdot support(Y)}{support(X, total\ FSA)} \quad (18)$$

Finally, using well known set operations, the final attribute can be calculated as follows:

support($X \rightarrow Y$) =

$$support(X) + support(Y) - confidence(X \rightarrow Y)\ support(X) \quad (19)$$

To sum up, for every rule $X \rightarrow Y$, the corresponding attributes can be calculated from an FSA using the following equations:

$$support(X \rightarrow Y) =$$

$$support(X) + support(Y) - confidence(X \rightarrow Y)\, support(X) \quad (20)$$

$$confidence(X \rightarrow Y) = \frac{support(X, local\ FSA) \cdot support(Y)}{support(X, total\ FSA)} \quad (21)$$

Based on the previous theoretical analysis, the `RuleCreator` algorithm is presented in Figure 10.

8. EXPERIMENTS

In order to validate the correctness of the PROTEAS methodology, the experimental procedures presented in [27] and [13] were used.

In the first case, the input was the 10 `PROSITE` protein families that were used in [27] (`PDOC00064`, `PDOC00154`, `PDOC00224`, `PDOC00271`, `PDOC00343`, `PDOC00561`, `PDOC00662`, `PDOC00670`, `PDOC00791`, `PDOC50007`), containing 585 protein sequences. Each time, a different portion of the initial data set was used for training, and the same experiment was performed multiple times. The average of the results is presented in Table 1.

Table 1 Comparative Experiments with [27][27]

Train Set (%) / Test Set (%)	PROTEAS	Decision Tree Algorithm [27]
2 (%) / 100 (%)	84.441 %	59.1 %
3 (%) / 100 (%)	83.232 %	71.7 %
5 (%) / 100 (%)	88.670 %	82.7 %
10 (%) / 100 (%)	96.223 %	94.0 %
20 (%) / 100 (%)	96.525 %	95.1 %
30 (%) / 100 (%)	99.546 %	96.5 %
40 (%) / 100 (%)	99.697 %	98.4 %
50 (%) / 100 (%)	99.546 %	99.0 %
60 (%) / 100 (%)	99.697 %	98.7 %
99 (%) / 100 (%)	99.697 %	99.8 %

In the second case [13], the portion of the initial data was kept constant, but the number of protein classes was varied. The 10 – class dataset is the one used in [27]. The 20 – and 30 – class datasets are the following:

20 – Class dataset: `PDOC00020`, `PDOC00023`, `PDOC00027`, `PDOC00335`, `PDOC00340`, `PDOC00344`, `PDOC00552`, `PDOC00561`, `PDOC00564`, `PDOC00567`, `PDOC00789`, `PDOC00790`, `PDOC00793`, `PDOC00800`, `PDOC00803`, `PDOC50001`, `PDOC50006` and `PDOC50017`

30 – Class dataset: `PDOC00010`, `PDOC00013`, `PDOC00021`, `PDOC00024`, `PDOC00027`, `PDOC00031`, `PDOC00061`, `PDOC00067`, `PDOC00072`, `PDOC00075`, `PDOC00160`, `PDOC00164`, `PDOC00166`, `PDOC00167`, `PDOC00171`, `PDOC00298`, `PDOC00300`, `PDOC00304`, `PDOC00305`, `PDOC00307`, `PDOC00310`, `PDOC00485`, `PDOC00486`, `PDOC00490`, `PDOC00495`, `PDOC00499`, `PDOC00722` and `PDOC00728`

Again, the experiments were performed multiple times, and the results shown in Table 2.

Table2. Comparative Experiments with [13]

Number of Classes	PROTEAS	Gen-Miner
10	93.713 %	80.130 %
19	94.310 %	84.268 %
28	75.983 %	74.490 %

A final experiment was performed on all available protein classes (Table 3). In this case, it was not possible to perform a comparative experiment, as we were not able to find another algorithm in literature that could accommodate all the protein families.

Table3. Experiment on all protein classes

Number of Classes	PROTEAS
1100	41.388 %

From the results, it is obvious that the classifier accuracy for the total of the protein families available in `PROSITE` is not satisfactory. However, it must be noted that the classifier was built using the simplest rules possible, i.e. one motif $X \rightarrow$ one class Y, owing to the high time complexity of the methodology. These simple rules cannot accurately describe the actual protein dataset, where each protein contains approximately five different motifs. Notwithstanding this fact, the PROTEAS methodology succeeds in accurately classifying almost half of the proteins in the dataset.

9. CONCLUSIONS

We have proposed a data mining methodology for interesting rule extraction, based on the construction of a stochastic deterministic finite state automaton. Using the protein sequences, a prefix tree acceptor is constructed, and consequently merged into an equivalent but more generalized finite state automaton. We have shown a new algorithm to extract set probabilities from an FSA, as opposed to sequence probabilities. Finally, we have applied the algorithm to create and evaluate classification rules for protein data.

Our results indicate that the PROTEAS methodology is efficient compared to other algorithms in the literature. However, it must be noted that all experiments were performed using the simple rules, i.e. one motif $X \rightarrow$ one class Y. This is the reason for the high classification error when the number of classes increases. The time complexity of the algorithm also indicates that for more complex rules the experimentation time would rise exponentially.

10. REFERENCES

[1] T.K. Attwood, M.D.R. Croning, D.R. Flower, A.P. Lewis, J.E. Mabey, P. Scordis, J. Selley, W. Wright, PRINT-S: the database formerly known as PRINTS, *Nucleic Acids Res. 28*, pp. 225–227, 2000.

[2] P.F. Baldi, S. Brunak, *Bioinformatics: A Machine Learning Approach*, The MIT Press, Cambridge, MA, 2001.

[3] A. Bateman, E. Birney, R. Durbin, S.R. Eddy, K.L. Howem, E.L.L. Sonnhammer, The Pfam protein families database, *Nucleic Acids Res. 28*, pp. 263–266, 2000.

[4] G. Bejerano, G. Yona, Variations on probabilistic suffix trees: statistical modeling and prediction of protein families, *Bioinformatics, Vol. 17 No. 1*, pp 23-43, 2001.

[5] C. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, New York, 1995.

[6] R.C. Carrasco, J. Oncina, Learning stochastic regular grammar by means of a state merging method, *Proc. The Second International Colloquium on Grammatical Inference* (ICGI '94), Alicante, Spain, Lecture Notes in Artificial Intelligence LNAI 862, pp. 139 – 152, Springer – Verlag, 1994.

[7] B. Cheng, J.G. Carbonell, J. Klein-Seetharaman, Protein Classification Based in Text Document Classification Techniques, *PROTEINS: Structure, Function, and Bioinformatics, Vol. 58*, pp. 955-970, 2005.

[8] M. Deshpande, G. Karypis, Evaluation of techniques for classifying biological sequences, *In 6th Pacific-Asia Conference on Knowledge Discover (PAKDD)*, pp. 417-431, 2002.

[9] S. Diplaris, G. Tsoumakas, P.A. Mitkas, I. Vlahavas, Protein classification with multiple algorithms, *In Proc. Of 10th Panhellenic Conference in Informatics*, Volos, Greece, 21-23 November, Springer-Verlag, LNCS 3746, pp. 446-456, 2005.

[10] R. Duad, P. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[11] L. Falquet, M. Pagni, P. Bucher, N. Hulo, C.J. Sigrist, K. Hofmann, A. Bairoch, The PROSITE database, its status in 2002, *Nucleic Acids Res. 30*, pp. 235–238, 2002.

[12] J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2001.

[13] G. Hatzidamianos, S. Diplaris, I. Athanasiadis, P.A. Mitkas, GenMiner: a Data Mining Tool for Protein Analysis, *9th Panhellenic Conference on Informatics*, pp. 346 – 360, November 21 – 23, 2003, Thessaloniki, Greece.

[14] P. Hingston, Using Finite State Automata for Sequence Mining, *Twenty – Fifth Australasian Computer Science Conference*, Melbourne, Australia, 105 – 110, 2001.

[15] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison – Wesley, 2001.

[16] B. Kero, L. Russell, S. Tsur, W.M. Shen, An Overview of Data Mining Technologies, *In: The KDD Workshop in the 4th International Conference on Deductive and Object-Oriented Databases*, Singapore, 1995.

[17] F.G. Leonardi, A generalization of the PST algorithm: modeling the sparse nature of protein sequences, *Bioinformatics, Vol. 22 No 11*, pp. 1302-1307, 2006.

[18] E.D. Levy, C.A. Ouzounis, W.R. Gilks, B. Audit, Probabilistic annotation of protein sequences based on functional classifications, *BMC Bioinformatics, Vol 6:302*, 2005.

[19] C. Makris, Y. Panagis, K. Perdikuri, E. Theodoridis, A. Tsakalidis, Algorithms for Protein Clustering, *In Proceeding of the 2nd International Greek Biotechnology Forum*, pp 38-44, July 1-3, 2005.

[20] G. Piatetsky-Shapiro, W.J. Frawley, *Knowledge Discovery in Databases*, MIT Press, 1992.

[21] C.P.Ponting, J.Schultz, F. Milpetz, P. Bork, SMART: identification and annotation of domains from signaling and extracellular protein sequences, *Nucleic Acids Res., 27*, pp. 229-232, 1999

[22] F. Psomopoulos, S. Diplaris, P.A. Mitkas, A Finite State Automata Based Technique for Protein Classification Rules Induction, *In Proceedings of the Second European Workshop on Data Mining and Text Mining for Bioinformatics*, pp. 54-60, ECML/PKDD 2004, Pisa, Italy, September 20-24, 2004.

[23] F.E. Psomopoulos, P.A Mitkas, A protein classification engine based on stochastic finite state automata, Lecture Series on Computer and Computational Sciences VSP/Brill, 4B (2005), pp. 1371-1374, *Proc. of the symp. 35: Computational Methods in Molecular Biology at the ICCMSE 2005*, Loutraki, Greece, 21-26 October, 2005.

[24] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.

[25] P. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining*, Addison-Wesley, 2006.

[26] S. Vinga, J. Almeida, Alignment-free sequence comparison-a review, *Bioinformatics, Vol. 19 No. 4*, pp. 513-523, 2003.

[27] D. Wang, X. Wang, V. Honavar, D. Dobbs, Data-driven generation of decision trees for motif-based assignment of protein sequences to functional families, *In: Proceedings of the Atlantic Symposium on Computational Biology*, Genome Information Systems & Technology, 2001.

[28] I.H. Witten, E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[29] C. Zhang, S. Zhang, *Association Rule Mining*, Springer, 2002.

[30] http://au.expasy.org/prosite/

[31] http://www.sanger.ac.uk/Software/Pfam/

[32] http://www.ebi.ac.uk/interpro/