A New Discrete Particle Swarm Algorithm Applied to Attribute Selection in a Bioinformatics Data Set

Elon S. Correa
Computing Laboratory
University of Kent
Canterbury, CT2 7NF, UK
E.S.Correa@kent.ac.uk

Alex A. Freitas
Computing Laboratory
University of Kent
Canterbury, CT2 7NF, UK
A.A.Freitas@kent.ac.uk

Colin G. Johnson
Computing Laboratory
University of Kent
Canterbury, CT2 7NF, UK
C.G.Johnson@kent.ac.uk

ABSTRACT

Many data mining applications involve the task of building a model for predictive classification. The goal of such a model is to classify examples (records or data instances) into classes or categories of the same type. The use of variables (attributes) not related to the classes can reduce the accuracy and reliability of a classification or prediction model. Superfluous variables can also increase the costs of building a model - particularly on large data sets. We propose a discrete Particle Swarm Optimization (PSO) algorithm designed for attribute selection. The proposed algorithm deals with discrete variables, and its population of candidate solutions contains particles of different sizes. The performance of this algorithm is compared with the performance of a standard binary PSO algorithm on the task of selecting attributes in a bioinformatics data set. The criteria used for comparison are: (1) maximizing predictive accuracy; and (2) finding the smallest subset of attributes.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning, induction*

General Terms

Algorithms, performance

Keywords

Particle swarm, optimization, Data Mining, attribute selection, Naive Bayes classifier, bioinformatics.

1. INTRODUCTION

Most of the particle swarm algorithms present in the literature deal only with continuous variables [1, 5, 10]. This is a significant limitation because many optimization problems are set in a space featuring discrete variables. Typical examples include problems which require the ordering or arranging of discrete variables, such as scheduling or routing problems [17]. Therefore, the design of particle swarm algorithms that deal with discrete variables is pertinent to this field of study.

We propose a discrete Particle Swarm Optimization (PSO) algorithm applied to attribute selection in Data Mining. We shall refer to this algorithm as the discrete Particle Swarm Optimization (DPSO) algorithm. The DPSO deals with discrete variables, and its population of candidate solutions contains particles of different sizes. Although the algorithm has been specifically designed for an attribute selection task, it is by no means limited to this kind of application. The DPSO algorithm may potentially be applied to other discrete optimization problems, such as facility location problems [2], with a few minor modifications.

Many data mining applications involve the task of building a model for predictive classification. The goal of such a model is to classify examples (records or data instances) into classes or categories of the same type. The use of variables (attributes) not related to the classes can reduce the accuracy and reliability of a classification or prediction model. Superfluous variables can also increase the costs of building a model - particularly on large data sets. The objective of attribute selection is to simplify a data set by reducing its dimensionality and identifying relevant underlying attributes without sacrificing predictive accuracy. By doing that, it also reduces redundancy in the information provided by the selected attributes. For a review of the attribute selection task using genetic algorithms see [4].

The DPSO algorithm was designed to tackle the data mining task of attribute selection. It differs from other traditional PSO algorithms because its particles do not represent points inside an n-dimensional Euclidean space (continuous case) or lattice (binary case) as in the standard PSO algorithms [9]. Instead, they represent a combination of selected attributes.

The paper is organized as follows. Section 2 briefly introduces PSO algorithms. Section 3 describes the standard binary PSO algorithm. Section 4 introduces the DPSO algorithm proposed in this paper for the task of attribute selection. Section 5 reports computational experiments, and describes the postsynaptic data set - the data set used in our experiments. It also includes a brief discussion of the results obtained. Section 6 presents conclusions of the work and points out future research directions. The next subsection specifies the notation used throughout this paper.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

1.1 Notation

A lowercase letter, e.g., x, denotes a random variable. An uppercase letter with an arrow over the letter, e.g., $\overrightarrow{X}$, denotes a vector of random variables. $\overrightarrow{X} = (x_1, x_2, ..., x_n)$ denotes an n-dimensional vector of random variables. Abusing the mathematical notation, we use $\overrightarrow{X} = \{x_1, x_2, ..., x_n\}$ (note the braces "{}") to represent a vector of random variables which is also a set of indices. $\overrightarrow{X} = \{x_1, x_2, ..., x_n\}$ is a set of indices in the mathematical sense of set. That is, there are no duplicated indices and there is no ordering among the indices $x_1, x_2, ..., x_n$. Given a candidate solution, say $\overrightarrow{X}(i)$, the symbol $f(\overrightarrow{X}(i))$, called the fitness function, represents a measurement of how well the solution $\overrightarrow{X}(i)$ solves the target problem. Subsection 5.2 describes how the measurement $f(\overrightarrow{X}(i))$ is computed in the present work.

2. A BRIEF INTRODUCTION TO PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) comprises a set of search techniques, inspired by the behavior of natural swarms, for solving optimization problems [9]. In PSO a potential solution to a problem is represented by a particle $\overrightarrow{X}(i) = (x_{(i,1)}, x_{(i,2)}, ..., x_{(i,n)})$ in an n-dimensional search space. The coordinates $x_{(i,d)}$ of these particles have a rate of change (velocity) $v_{(i,d)}$, $d = 1, 2, ..., n$. Every particle keeps a record of the best position that it has ever visited. Such a record is called the particle's previous best position and denoted by $\overrightarrow{B}(i)$. The global best position attained by any particle so far is also recorded and stored in a particle denoted by $\overrightarrow{G}$. An iteration comprises evaluation of each particle, then stochastic adjustment of $v_{(i,d)}$ in the direction of particle $\overrightarrow{X}(i)$'s previous best position and the previous best position of any particle in the neighborhood [8]. There is much variety in the neighborhood topology used in PSO, but quite often *gbest* or *lbest* topologies are used. In the *gbest* topology every particle has only the global best particle $\overrightarrow{G}$ as its neighbor. In the *lbest* topology, usually, each particle has a number of other particles to its right and left as neighbors. For a review of the neighborhood topologies used in PSO the reader is referred to [7, 9].

Generally speaking, the set of rules that govern PSO are: evaluate, compare and imitate. The evaluation phase measures how well each particle (candidate solution) solves the problem at hand. The comparison phase identifies the best particles. The imitation phase produces new particles based on some of the best particles previously found. These three phases are repeated until a given stopping criterion is met. The objective is to find the particle that best solves the target problem.

Important concepts in PSO are velocity and neighborhood topology. Each particle, $\overrightarrow{X}(i)$, is associated with a velocity vector. This velocity vector is updated at every generation. The updated velocity vector is then used to generate a new particle $\overrightarrow{X}(i)$. The neighborhood topology defines how other particles in the swarm, such as $\overrightarrow{B}(i)$ and $\overrightarrow{G}$, interact with $\overrightarrow{X}(i)$ to modify its respective velocity vector and, consequently, its position as well.

3. THE STANDARD BINARY PSO ALGORITHM

The standard binary version of the PSO algorithm [9] works as follows. Potential solutions (particles) to the target problem are encoded as fixed length binary strings; i.e., $\overrightarrow{X}(i) = (x_{(i,1)}, x_{(i,2)}, ..., x_{(i,n)})$, where $x_{(i,j)} \in \{0, 1\}$, $i = 1, 2, ..., N$ and $j = 1, 2, ..., n$. Given a list of attributes $A = (A_1, A_2, ..., A_n)$, the first element of $\overrightarrow{X}(i)$, from the left to the right hand side, corresponds to the first attribute "A_1", the second to the second attribute "A_2", and so forth. A value of 0 on the site associated to an attribute signifies that the respective attribute is not selected. A value of 1 means that it is selected. For example, given the list of attributes $A = (A_1, A_2, A_3, A_4, A_5)$ and $N = 4$, a swarm could look like this:

$$\begin{array}{|l|}\hline \overrightarrow{X}(1) = (0, 1, 1, 0, 1) \\ \overrightarrow{X}(2) = (1, 0, 0, 1, 0) \\ \overrightarrow{X}(3) = (0, 1, 0, 1, 1) \\ \overrightarrow{X}(4) = (1, 1, 1, 0, 0) \\ \hline \end{array}$$

In this example, particle $\overrightarrow{X}(1) = (0, 1, 1, 0, 1)$ represents a candidate solution where attributes A_2, A_3 and A_5 are the only attributes selected.

3.1 The initial population for the standard binary PSO algorithm

For the initial population, N binary strings of length n are randomly generated. Each particle $\overrightarrow{X}(i)$ is independently generated as follows. For every position $x_{(i,d)}$ of $\overrightarrow{X}(i)$ a uniform random number φ is drawn on the interval (0, 1). If $\varphi < 0.5$, then $x_{(i,d)} = 1$, otherwise $x_{(i,d)} = 0$.

3.2 Updating the records

At the beginning, the previous best position of $\overrightarrow{X}(i)$, denoted by $\overrightarrow{B}(i)$, is empty. Therefore, once the initial particle $\overrightarrow{X}(i)$ is generated, $\overrightarrow{B}(i)$ is set to $\overrightarrow{B}(i) = \overrightarrow{X}(i)$. After that, every time that $\overrightarrow{X}(i)$ is updated, $\overrightarrow{B}(i)$ is also updated if $f(\overrightarrow{X}(i))$ is better than $f(\overrightarrow{B}(i))$. Otherwise, $\overrightarrow{B}(i)$ remains as it is. A similar process is used to update the global best position $\overrightarrow{G}$. At the beginning, $\overrightarrow{G}$ is also empty. Therefore, once all the $\overrightarrow{B}(i)$ have been determined, $\overrightarrow{G}$ is set to the fittest $\overrightarrow{B}(i)$ previously computed. After that, $\overrightarrow{G}$ is updated if the fittest $f(\overrightarrow{B}(i))$ in the swarm is better than $f(\overrightarrow{G}(i))$. And, in that case, $f(\overrightarrow{G}(i))$ is set to $f(\overrightarrow{G}(i)) =$ fittest $f(\overrightarrow{B}(i))$. Otherwise, $\overrightarrow{G}$ remains as it is.

3.3 Updating the velocities for the standard binary PSO algorithm

Every particle i is associated to a unique vector of velocities $V(i) = (v_{(i,1)}, v_{(i,2)}, ..., v_{(i,n)})$. The elements $v_{(i,d)}$ in $V(i)$ determine the rate of change of each respective coordinate $x_{(i,d)}$ in $\overrightarrow{X}(i)$, $d = 1, 2, ..., n$. Each element $v_{(i,d)} \in V(i)$ is updated according to the equation:

$$v_{(i,d)} = w\, v_{(i,d)} + \varphi_1(b_{(i,d)} - x_{(i,d)}) + \varphi_2(g_{(d)} - x_{(i,d)}), \quad (1)$$

where w $(0 < w < 1)$, called the inertia weight, is a constant value chosen by the user. Equation 1 is a standard equation used in PSO algorithms to update the velocities [6, 14]. Note

that $x_{(i,d)}$ is the d^{th} component of $\overrightarrow{X}(i)$; $b_{(i,d)}$ is the d^{th} component of $\overrightarrow{B}(i)$; $g_{(d)}$ is the d^{th} component of $\overrightarrow{G}$ and $d = 1, 2, ..., n$. The factors φ_1 and φ_2 are uniform random numbers independently generated in the interval (0, 1).

3.4 Sampling new particles for the standard binary PSO algorithm

New particles are then sampled as follows. For each particle i and each dimension d, the value of the new coordinate $x_{(i,d)} \in \overrightarrow{X}(i)$ can be either 0 or 1. The decision of whether $x_{(i,d)}$ will be 0 or 1 is based on its respective velocity $v_{(i,d)} \in V(i)$ and is given by the following equation:

$$x_{(i,d)} = \begin{cases} 1, & \text{if(rand} < S(v_{(i,d)})) \\ 0, & \text{otherwise;} \end{cases} \quad (2)$$

where $0 \leq \text{rand} \leq 1$ is a uniform random number and $S(v_{(i,d)}) = \dfrac{1}{1+\exp(-v_{(i,d)})}$ is the sigmoid function. Equation 2 is a standard equation used to sample new particles in the binary PSO algorithm [9]. Note that the lower the value of $v_{(i,d)}$ the more likely the value of $x_{(i,d)}$ will be 0. By contrast, the higher the value of $v_{(i,d)}$ the more likely the value of $x_{(i,d)}$ will be 1. The next section presents the DPSO algorithm prosed in this paper.

4. THE PROPOSED DISCRETE PSO ALGORITHM

This algorithm deals with discrete variables (attributes) and its population of candidate solutions contains particles of different sizes. Potential solutions to the optimization problem at hand are represented by a swarm of particles. There are N particles in a swarm. The length of each particle may vary from 1 to n, where n is the number of attributes of the problem. Each particle $\overrightarrow{X}(i)$ keeps a record of the best position it has ever attained. This information is stored in a separated particle labeled as $\overrightarrow{B}(i)$. The swarm also keeps a record of the global best position ever attained by any particle in the swarm. This information is also stored in a separated particle labeled $\overrightarrow{G}$. Note that $\overrightarrow{G}$ is equal to the best $\overrightarrow{B}(i)$ present in the swarm.

4.1 Encoding of the particles for the proposed DPSO algorithm

Each attribute is identified by a unique positive integer number, or index. These numbers, indices, vary from 1 to n. A particle is a subset of non-ordered indices without repetition, e.g., $\overrightarrow{X}(i) = \{2, 4, 18, 1\}$. For example, given the list of attributes (A_1, A_2, A_3, A_4, A_5) and $N = 4$, a swarm could look like this:

$$\begin{array}{|l|}\hline \overrightarrow{X}(1) = \{4, 1, 2\} \\ \overrightarrow{X}(2) = \{5\} \\ \overrightarrow{X}(3) = \{2, 1\} \\ \overrightarrow{X}(4) = \{1, 3, 2, 5, 4\} \\ \hline \end{array}$$

Note that in the particle $\overrightarrow{X}(1) = \{4, 1, 2\}$, 4 corresponds to attribute A_4, 1 to A_1 and 2 to A_2. Therefore, $\overrightarrow{X}(1) = \{4, 1, 2\}$ represents a candidate solution where the attributes A_4, A_1, and A_2 have been selected.

4.2 The initial population for the proposed DPSO algorithm

The initial population of particles is generated as follows. At the beginning, an integer random number determines the number of attribute indices, or length, that a particle will have. This number is uniformly generated between 1 and n, inclusive, individually for every single particle. Particles bear their randomly chosen size throughout the execution of the algorithm. Once the dimensionality is known, the actual particle is generated with as many attribute indices as the previously chosen random number that corresponds to its size. For instance, if the uniform random number $k \in \{1, 2, 3, ..., n\}$ that determines the length of a particle is drawn as $k = 2$, a particle that contains exactly 2 attribute indices is generated. Those indices are also randomly chosen from the sequence $I = \{1, 2, 3, ..., n\}$, one at a time, and without replacement. It means that there will be no repeated indices on the configuration of any single particle. Algorithm 1 shows a pseudocode for the encoding of a discrete particle. Note that the particles are completely generated one-by-one. After the initial population is generated,

Algorithm 1 Encoding of a discrete particle

Require: i, j, ℓ, n, N, $\overrightarrow{X}(i)$, $I = \{1, 2, ..., n\}$ and $1 \leq rand \leq n$, where $rand$ is a uniform random number, $rand \in \{1, 2, ..., n\}$.

```
for i = 1 to N
    ℓ = rand
    X(i) = ∅
    I = {1, 2, ..., n}
  for j = 1 to ℓ
    Randomly select an attribute (index) from I
    Insert the selected attribute (index) in X(i)
    I = I - X(i), (Recall that X(i) is also a set.)
  end for
end for
```

the information in $\overrightarrow{B}(i)$ and $\overrightarrow{G}$ is then updated exactly as described in Subsection 3.2.

4.3 Velocities = proportional likelihoods

The DPSO algorithm does not use a vector of velocities as the standard PSO algorithm does. It works with proportional likelihoods instead. Arguably, the notion of proportional likelihood used in the DPSO algorithm and the notion of velocity used in the standard PSO are somewhat similar. Every particle is associated with a 2-by-n array of proportional likelihoods, where 2 is the number of rows in this array and n is the number of columns. A generic proportional likelihood array looks like this:

$V(i) = \begin{pmatrix} \text{proportional likelihood row} \\ \text{attribute index row} \end{pmatrix}$. Each of the n elements in the first row of $V(i)$ represents the proportional likelihood that an attribute be selected. The second row of $V(i)$ shows the indices of the attributes associated with the respective proportional likelihoods. There is a one-to-one correspondence between the columns of this array and the attributes of the problem domain. At the beginning, all elements in the first row of $V(i)$ are set to 1, e.g., $V(i) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$. After the initial population of particles is generated, this array is always updated before a new configuration for the

particle associated to it is generated. The updating process is based on $\overrightarrow{X}(i)$, $\overrightarrow{B}(i)$ and $\overrightarrow{G}$ and works as follows. In addition to $\overrightarrow{X}(i)$, $\overrightarrow{B}(i)$ and $\overrightarrow{G}$, three constant updating factors, namely, α, β and γ, are also used to update the proportional likelihoods $v_{(i,d)}$. These factors determine the strength of the contribution of $\overrightarrow{X}(i)$, $\overrightarrow{B}(i)$ and $\overrightarrow{G}$ to the adjustment of every coordinate $v_{(i,d)} \in V(i)$. Note that α, β and γ are parameters chosen by the user. The contribution of these parameters to the updating of $v_{(i,d)}$ is as follows. All indices present in $\overrightarrow{X}(i)$ have their correspondent proportional likelihood increased by α. In addition to that, all indices present in $\overrightarrow{B}(i)$ have their correspondent proportional likelihood increased by β. The same for $\overrightarrow{G}$ for which the proportional likelihoods are increased by γ. For instance, given $n = 5$, $\alpha = 0.10$, $\beta = 0.12$, $\gamma = 0.14$, $\overrightarrow{X}(i) = \{2, 3, 4\}$, $\overrightarrow{B}(i) = \{3, 5, 2\}$, $\overrightarrow{G} = \{5, 2\}$ and $V(i) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$, the updated $V(i)$ would be: $V(i) = \begin{pmatrix} 1 & 1+\alpha+\beta+\gamma & 1+\alpha+\beta & 1+\alpha & 1+\beta+\gamma \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$. Note that index 1 is not present in $\overrightarrow{X}(i)$, $\overrightarrow{B}(i)$ or $\overrightarrow{G}$. Therefore, the proportional likelihood of attribute 1 in $V(i)$ remains as it is. This new updated array replaces the old one and will be used to generate a new configuration to the particle associated to it as follows.

4.4 Sampling new particles for the proposed DPSO algorithm

The proportional likelihood array $V(i)$ is then used to sample a new instance of particle $\overrightarrow{X}(i)$ - that is, the particle associated to it. First, every element of the first row of the array $V(i)$ is multiplied by a uniform random number between 0 and 1. A new random number is drawn for every single multiplication performed. To illustrate, suppose that $V(i) = \begin{pmatrix} 1 & 1.36 & 1.22 & 1.1 & 1.26 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$. The multiplied proportional likelihood array would be: $V(i) = \begin{pmatrix} 1 \times \varphi_1 & 1.36 \times \varphi_2 & 1.22 \times \varphi_3 & 1.1 \times \varphi_4 & 1.26 \times \varphi_5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$, where φ_1, ..., φ_5 are uniform random numbers independently drawn on the interval (0, 1). Suppose that the multiplied array $V(i)$ looks like this: $V(i) = \begin{pmatrix} 0.11 & 0.86 & 0.57 & 0.62 & 1.09 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$. The new particle is then defined by ranking the columns in $V(i)$ by the values in its first row. That is, the elements in the first row of the array are ranked in a decreasing order of value and the indices of the attributes (in the second row of $V(i)$) follow their respective proportional likelihoods. For example, ranking $V(i) = \begin{pmatrix} 0.11 & 0.86 & 0.57 & 0.62 & 1.09 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$, we would obtain $V(i) = \begin{pmatrix} 1.09 & 0.86 & 0.62 & 0.57 & 0.11 \\ 5 & 2 & 4 & 3 & 1 \end{pmatrix}$. After ranking the array $V(i)$, the first k indices (in the second row of $V(i)$), from left to right, are selected to compose the new particle. The constant k represents the length of the particle $\overrightarrow{X}(i)$, the particle associated to the ranked array $V(i)$. Thus, if particle i, the particle associated to the array $V(i) = \begin{pmatrix} 1.09 & 0.86 & 0.62 & 0.57 & 0.11 \\ 5 & 2 & 4 & 3 & 1 \end{pmatrix}$, has length 3, the first 3 indices from the second row of $V(i)$ would be selected to compose the particle. Based on the array $V(i)$ given above, if $k = 3$ (that is, $\overrightarrow{X}(i) = \{*, *, *\}$) the indices (attributes) 5, 2 and 4 would be selected to compose the new particle, i.e., $\overrightarrow{X}(i) = \{5, 2, 4\}$. Note that indices that have a higher proportional likelihood are, on average, more likely to be selected.

The updating of $\overrightarrow{X}(i)$, $\overrightarrow{B}(i)$ and $\overrightarrow{G}$ is identical to what is described in Subsection 3.2.

5. EXPERIMENTS

In this section, we report and discuss computational experiments. The quality of a candidate solution (fitness) is computed by the well-known Naive Bayes classifier [11]. The Naive Bayes classifier uses a probabilistic approach to assign each example (record) of the data set to a possible class. In our application, it assigns a record (protein) of the data set to one of the classes, negative or positive. A Naive Bayes classifier assumes that all attributes are probabilistic independent of one another.

5.1 Postsynaptic data set

This section presents the bioinformatics data set used in the present work for attribute selection. A synapse is a connection between two neurons: presynaptic and postsynaptic. The first is usually the sender of a "signal", such as the release of chemicals, while the second is the receiver. A postsynaptic receptor is a sensor on the surface of a neuron. It captures messenger molecules from the nervous system, neurotransmitters, and thereby functions in transmitting information from one neuron to another [15]. The data set used in this paper is called the postsynaptic data set. It has been recently created and mined for the first time in [13]. The data set contains 4303 records of proteins. These proteins belong to either one of the following two classes: positive or negative. Proteins that belong to the positive class have postsynaptic activity. Proteins that belong to the negative class do not have postsynaptic activity. From the 4303 proteins on the data set, 260 belong to the positive class and 4043 to the negative class.

This data set is a particularly interesting case study for evaluating the proposed DPSO algorithm for two reasons. First, postsynaptic proteins are involved in the nervous system. Predicting postsynaptic activity is potentially useful for understanding several diseases of the nervous system. Second, this data set has many attributes what makes the attribute selection task challenging. More precisely, each protein has 443 PROSITE patterns, or attributes.

PROSITE is a database of protein families and domains. It is based on the observation that, while there is a huge number of different proteins, most of them can be grouped, on the basis of similarities in their sequences, into a limited number of families (a protein consists of a sequence of amino acids). PROSITE patterns are small regions within a protein that present a high sequence similarity when compared to other proteins. In our data set the absence of a given PROSITE pattern is indicated by a value of 0 for the attribute corresponding to that PROSITE pattern. The presence of it is indicated by a value of 1 for that same attribute.

5.2 Experimental Methodology

The fitness function $f(\overrightarrow{X}(i))$ of a any particle i is computed as follows. $f(\overrightarrow{X}(i))$ is equal to the predictive accuracy achieved by a Naive Bayes classifier on the postsynaptic data set and using only the attributes present in $\overrightarrow{X}(i)$. The objective is to find the smallest subset of attributes (PROSITE patterns) with which it is possible to classify the proteins on the data set as belonging to one of the classes (positive or negative) with an acceptable accuracy. We define the accuracy as acceptable if it is equal to or better than the one obtained by the classification performed considering all the 443 original attributes. Note that this is a naive and particular definition of acceptable accuracy. We chose this definition because it suits the purpose of our experiments - to compare the performance of the standard binary PSO and the DPSO algorithms in the postsynaptic data set. As a rule, the definition of acceptable accuracy is problem dependent and should take into account prior knowledge of the target problem - when available. In fact, in many real-world applications, minimizing the number of selected attributes while maximizing classification accuracy are conflicting tasks.

The measurement of $f(\overrightarrow{X}(i))$ in this paper follows what in Data Mining is called a wrapper approach. The wrapper approach searches for an optimal attribute subset tailored to a particular algorithm, such as the Naive Bayes classifier. For more information on wrapper and other attribute selection approaches see [18].

The computational experiments involved a 10-fold cross-validation method [18]. First, the 4303 records in the postsynaptic data set were divided into 10 almost equally sized folds. There are three folds containing 431 records each one and seven folds containing 430 records each one. The folds were randomly generated but under the following regulation. The proportion of positive and negative classes in every single fold must be similar to the one found in the original data set containing all the 4303 records. This is known as stratified cross-validation. Each of the 10 folds is used once as test set and the remaining of the data set is used as training set. Out of the 9 folds in the training set, one is reserved to be used as a validation set. The Naive Bayes classifier uses the remaining 8 folds to compute the probabilities required to classify new examples. Once those probabilities have been computed, the Naive Bayes classifier classifies the examples in the validation set. The accuracy of this classification on the validation set is the value of the fitness function $f(\overrightarrow{X}(i))$. After the run of the PSO algorithm is completed, the 9 folds are merged into a full training set. The Naive Bayes classifier is then trained again on this full training set (9 merged folds), and the probabilities computed in this final, full training set are used to classify examples in the test set (the 10th reserved fold), which was never accessed during the run of the PSO algorithm.

In each of the 10 iterations of the cross-validation procedure, the predictive accuracy of the classification is assessed by 3 different methods:

(1) **Using all the 443 original attributes**: all possible attributes are used by the Naive Bayes classifier.

(2) **Standard binary PSO algorithm**: only the attributes selected by the best particle found by the binary PSO algorithm are used by the Naive Bayes classifier.

(3) **Proposed DPSO algorithm**: only the attributes selected by the best particle found by the DPSO algorithm are used by the Naive Bayes classifier.

As the standard binary PSO and the DPSO algorithms are stochastic algorithms, 30 independent runs for each algorithm were performed for every single fold. The results obtained, averaged over 30 runs, are reported in Table 1. Since the Naive Bayes classifier is deterministic, only one run is performed for the classification using all the 443 original attributes. The average number of attributes selected by the attribute selection algorithms has always been rounded to the nearest integer. The population size used for both algorithms is 200 and the search stops after 20,000 fitness evaluations (or 100 iterations). The binary PSO algorithm uses a inertia weight value of 0.8 (i.e., $w = 0.8$). The choice of the value of this parameter was based on the work presented in [16]. Other choices of parameter values were $\alpha = 0.10$, $\beta = 0.12$ and $\gamma = 0.14$. These values were empirically determined in our preliminary experiments; but we make no claim that these are optimal values. Parameter optimization is a topic for future research.

The measurement of the predictive accuracy rate of a model should be a reliable estimate of how well that model classifies the test examples (unseen during the training phase) on the target problem. In Data Mining, typically, the equation:

$$\textit{Standard accuracy rate} = \frac{TP + TN}{TP + FP + FN + TN} \qquad (3)$$

is used to assess the accuracy rate of a classifier (see the definition of TP, TN, etc. below). Nevertheless, if the class distribution is highly unbalanced, which is the case with the postsynaptic data set, Equation 3 is an ineffective way of measuring the accuracy rate of a model. For instance, on a data set in which 90% of the examples belong to the positive class and 10% to the negative class, it would be easy to maximize Equation 3 by simply predicting always the majority class. Therefore, on our experiments we use a more demanding measurement for the accuracy rate of a classification model. It has also been used before in [13], the paper in which the postsynaptic data set was used for the first time. This measurement is given by the equation:

$$\textit{Predictive accuracy rate} = TPR \times TNR, \qquad (4)$$

where, $TPR = \frac{TP}{TP + FN}$ and $TNR = \frac{TN}{TN + FP}$.

Note that if any of the quantities TPR or TNR is zero, the value returned by Equation 4 is also zero. Also note that TP (true positives) is the number of records correctly classified as positive class and FP (false positives) is the number of records incorrectly classified as positive class. TN (true negatives) is the number of records correctly classified as negative class and FN (false negatives) is the number of records incorrectly classified as negative class.

5.3 Discussion

Analyzing the predictive accuracy values ($TPR \times TNR$) shown in Table 1 we see that, on average, the binary PSO and the DPSO algorithms obtained a higher predictive accuracy value than the classification performed using all the 443 original attributes. The only exception being for fold number 7. The results obtained for the average predictive accuracy values suggest that the predictive accuracy of the

Table 1: The postsynaptic data set. Results of the classification performed by a Naive Bayes classifier using: (1) all possible attributes; (2) only the attributes given by the best solution found by the binary PSO algorithm and (3) only the attributes given by the best solution found by the DPSO algorithm. The results shown for the binary PSO and DPSO algorithms are averaged over 30 independent runs. The average number of attributes selected has always been rounded to the nearest integer.

		Using all the 443 original attributes			Binary PSO algorithm				Proposed DPSO algorithm			
Fold	No. of instances	TPR	TNR	$TPR \times TNR$	TPR	TNR	$TPR \times TNR$	No. of attributes selected	TPR	TNR	$TPR \times TNR$	No. of attributes selected
1	431	0.73	0.99	0.72	0.73	0.99	0.72	32	0.73	1.00	0.73	14
2	431	0.85	0.99	0.84	0.88	0.99	0.87	17	0.90	1.00	0.90	10
3	431	0.75	0.99	0.74	0.75	0.99	0.74	24	0.75	0.99	0.74	11
4	430	0.00	1.00	0.00	0.71	0.98	0.70	29	0.75	0.99	0.74	12
5	430	0.52	0.99	0.51	0.52	1.00	0.52	25	0.52	1.00	0.52	13
6	430	0.80	0.99	0.79	0.80	0.99	0.79	35	0.80	1.00	0.80	17
7	430	0.72	0.99	0.71	0.69	0.99	0.68	34	0.68	1.00	0.68	15
8	430	0.81	0.99	0.80	0.81	0.99	0.80	28	0.81	0.99	0.80	11
9	430	0.69	0.99	0.68	0.69	0.99	0.68	31	0.69	0.99	0.68	14
10	430	0.75	0.99	0.74	0.77	0.99	0.76	16	0.79	1.00	0.79	10
Average		0.66	0.99	0.66	0.74	0.99	0.73	27.10	0.74	1.00	0.74	12.70
Std. error		0.08	0.00	0.08	0.03	0.00	0.03	2.09	0.03	0.00	0.03	0.73

classification of the proteins on the data set analyzed improves when using not all but only a small subset of relevant attributes to perform the classification.

To assess the performance of the binary PSO algorithm and the DPSO algorithm we have considered two criteria: (1) maximizing predictive accuracy; and (2) finding the smallest subset of attributes. Comparing the first criterion, accuracy, we note that the DPSO algorithm did slightly better than the binary PSO algorithm. However, the difference is negligible (the average predictive accuracy value for the binary PSO algorithm was equal to 0.73, whereas for the DPSO algorithm it was equal to 0.74). The discriminating factor between the performance of these algorithms seems to be on the second criteria, finding the smallest subset of attributes. For all the 10 folds the DPSO algorithm selected a smaller subset of attributes. The difference between the average number of attributes selected by the binary PSO algorithm, which was equal to 27.10, and by the DPSO algorithm, which was equal to 12.70, clearly indicates that the second algorithm tends to finding a smaller subset of attributes than the first one does. One of the reasons for that seems to be the way in which the initial population of particles is generated for each algorithm. For the binary PSO algorithm the average number of attributes selected (which is the number of ones in the particle's configuration) on the particles of the initial population follows a binomial distribution [3]. The binomial distribution is given by:

$$P(k|n) = \frac{n!}{k!(n-k)!}\, p^k\, (1-p)^{n-k}, \tag{5}$$

where $P(k|n)$ represents the probability of obtaining exactly k successes out of n Bernoulli trials. Translating it to the binary PSO algorithm, n represents the total number of attributes ($n = 443$) of the problem and k the number of selected attributes (the number of ones in the particle's configuration). The result of each Bernoulli trial is true with probability p and false with probability $q = (1 - p)$ [12]. Note that for the initial particles generated by the binary PSO algorithm, the probability that an attribute be selected (success) is $p = 0.5$ and the probability of it not be selected is $q = (1 - 0.5) = 0.5$. Therefore, the probability of a particle with k attributes being generated is equal to:

$$\begin{aligned} P(k|443) &= \frac{443!}{k!(443-k)!}\, 0.5^k\, (1-0.5)^{443-k} \\ &= \frac{443!}{k!(443-k)!}\, 0.5^{443}. \end{aligned} \tag{6}$$

From Equation 6 we conclude that particles that have nearly 221 attributes selected ($k \approx 221$) are more likely to be generated than particles with any other number of selected attributes. The more the value k distances from 221 towards 1 or towards 443, the less likely a particle with length equal to k will be generated. Therefore, the length of the particles in the initial population of the binary PSO algorithm will be, on average, concentrated around $\frac{n}{2}$. Recall that in our application $\frac{n}{2} \approx 221$. We carried out an experiment that seems to corroborate what has been said. We generated an initial population of particles for the problem addressed in this paper exactly as described in Subsection 3.1. Recall that in this case $n = 443$ and $1 \leq k \leq 443$. This population contains 10,000 randomly generated particles. We then recorded the length of every particle generated. Figure 1 shows the number of occurrences of every particle's size (from 1 to 443) observed on those 10,000 randomly generated particles. For the DPSO algorithm the average number of attributes selected on the particles of the initial population is given by the equation:

$$P(k) = \frac{1}{n}, \;\; \forall\, k \in \{1, 2, ..., n\}, \tag{7}$$

where n represents the total number of attributes ($n = 443$) of the problem and k the number of selected attributes, or the length of the particle. From Equation 7 we conclude that the probability of a particle of length k being generated is equal to $\frac{1}{n}$ for all $k \in \{1, 2, ..., n\}$. Therefore, the length of the particles in the initial population of the DPSO

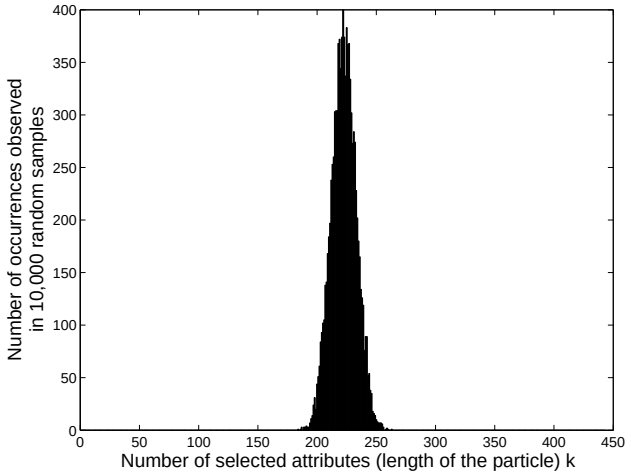

Figure 1: Binary PSO algorithm: an initial population generated at random. The population contains 10,000 particles which were generated as described in Subsection 3.1 for $n = 443$ and $1 \leq k \leq 443$.

algorithm tends to be uniformly distributed between 1 and n. We also generated an initial population of particles for the problem addressed in this paper exactly as described in Section 4. This population contains 10,000 randomly generated particles. We then recorded the length of every particle generated. Figure 2 shows the number of occurrences of every particle's size (from 1 to 443) observed on those 10,000 randomly generated particles. The fact that the DPSO algorithm has many more small sized particles in the initial population, by comparison with the standard binary PSO algorithm, seems to help the former to obtain smaller sets of selected attributes than the latter - as shown in the results of Table 1. Another trend observed in the results was

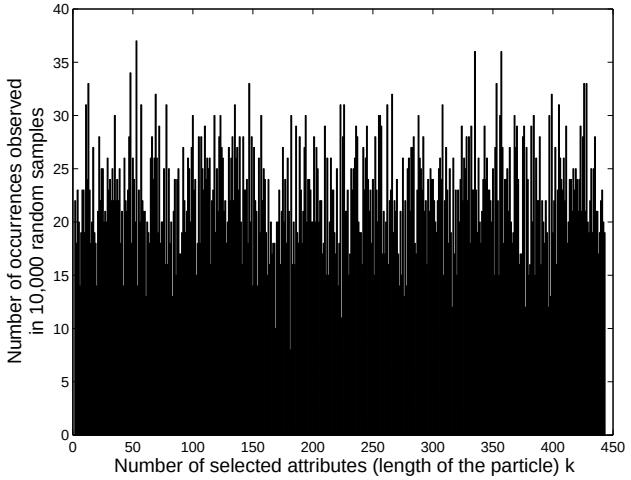

Figure 2: DPSO algorithm: an initial population generated at random. The population contains 10,000 particles which were generated as described in section 4 for $n = 443$ and $1 \leq k \leq 443$.

the high frequency of some attributes on the best particles found at each run of the attribute selection algorithms. We recorded the best particle found by each algorithm (i.e., the binary PSO and the DPSO algorithms) on each of the 30 runs and for every one of the 10 folds. We then computed the frequency of the attributes selected on the $10 \times 30 = 300$ best particles found by the standard binary PSO algorithm. The same was done for the proposed DPSO algorithm. The following 10 attributes have been selected, by both algorithms, in more than 85% of their respective 300 best particles found: A_{134}, A_{162}, A_{186}, A_{320}, A_{321}, A_{333}, A_{342}, A_{351}, A_{352} and A_{353}. The names of the PROSITE patterns that correspond to these attributes are shown in Table 2. The information was obtained from the web site of the European Bioinformatics Institute, UniProtKB/Swiss-Prot. Address: http://www.ebi.ac.uk/swissprot/.

Table 2: PROSITE patterns selected in more than 85% of the runs performed by the attribute selection algorithms.

Attribute/ PROSITE pattern ID	Name
A_{134}/ps00086	Cytochrome P450
A_{162}/ps00904	Protein prenyltransferase, alpha subunit
A_{186}/ps00856	Guanylate kinase
A_{320}/ps00713	Sodium: dicarboxylate symporter
A_{321}/ps00714	Sodium: dicarboxylate symporter
A_{333}/ps00405	43 kDa postsynaptic protein
A_{342}/ps00410	Dynamin
A_{351}/ps00232	Cadherin
A_{352}/ps00236	Neurotransmitter-gated ion-channel
A_{353}/ps00237	Rhodopsin-like GPCR superfamily

This information may be useful for a biologist. It suggests that those 10 PROSITE patterns are the most relevant ones for the classification of the proteins in the given data set.

6. CONCLUSIONS

The computational results indicate that the use of variables apparently unrelated to the class attribute tends to reduce the accuracy and reliability of a classification model on the postsynaptic data set. Using fewer attributes, the binary PSO and the DPSO algorithms obtained, on average, a better predictive accuracy than the classification performed using all the 443 original attributes. The overall results also indicate that, in the data set used in this paper, the DPSO algorithm performed as well as or better than the binary PSO algorithm. These two algorithms obtained a similar level of predictive accuracy. However, the DPSO algorithm clearly tends to find smaller subsets of attributes than the standard binary PSO algorithm does - as shown in Table 1. Perhaps, a partial reason for this difference is the way in which the initial population is generated for each algorithm. For the standard binary PSO algorithm the average number of attributes in the particles at the initial population is roughly $\frac{n}{2}$, where n is the number of attributes of the data set being mined. For the DPSO algorithm the length of the particles in the initial population tends to be uniformly distributed between 1 and n.

In future research we intend to investigate to what extent the apparent advantage in the performance of the DPSO

algorithm over the binary PSO algorithm is because of the way in which the initial populations are generated. This investigation will require the application of the algorithms to a variety of different test problems. We also intend to improve the DPSO algorithm by allowing the particles to vary in length during the execution of the algorithm. Perhaps, a mutation-like operator can be implemented so that the length of a particle may increase or decrease at random. Another idea is to generated the initial population such that it contains one instance of particle of every possible size for a particle from 1 to n, where n is the number of variables of the target problem. Parameter optimization is also a topic for future research.

7. ACKNOWLEDGMENTS

Thanks to Gisele L. Pappa for kindly providing us with the postsynaptic data set.

8. REFERENCES

[1] T. Blackwell and J. Branke. Multi-swarm optimization in dynamic environments. In *Lecture Notes in Computer Science*, volume 3005, pages 489–500. Springer-Verlag, 2004.

[2] E. S. Correa, M. T. Steiner, A. A. Freitas, and C. Carnieri. Using a genetic algorithm for solving a capacity p-median problem. *Numerical Algorithms*, 35:373–388, 2004.

[3] D. Freedman, R. Pisani, and R. Purves. *Statistics*. W. W. Norton & Company, 3rd edition, September 1997.

[4] A. A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, October 2002.

[5] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer for dynamic optimization problems. In *Evoworkshops 2004: 1st European Workshop on Evolutionary Algorithms in Stochastic and Dynamic Environments*, pages 513–524, Coimbra, Portugal, 2004. Springer-Verlag.

[6] G. Kendall and Y. Su. A particle swarm optimisation approach in the construction of optimal risky portfolios. In *Proceedings of the 23rd IASTED International Multi-Conference on Applied Informatics*, pages 140–145, 2005. Artificial intelligence and applications.

[7] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, pages 1931–1938, Piscataway, NJ, USA, 1999. IEEE Press.

[8] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 Conference on Systems, Man, and Cybernetics*, pages 4104–4109, Piscataway, NJ, USA, 1997. IEEE.

[9] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[10] M. Løvbjerg and T. Krink. Extending particle swarm optimisers with self-organized criticality. In D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1588–1593. IEEE Press, 2002.

[11] T. M. Mitchell. *Machine Learning*. McGraw-Hill, August 1997.

[12] A. Papoulis and S. U. Pillai. *Probability, Random Variables and Stochastic Processes With Errata Sheet*. McGraw-Hill, 1st edition, December 2001.

[13] G. L. Pappa, A. J. Baines, and A. A. Freitas. Predicting post-synaptic activity in proteins with data mining. *Bioinformatics*, 21(2):ii19–ii25, 2005.

[14] R. Poli, C. D. Chio, and W. B. Langdon. Exploring extended particle swarms: a genetic programming approach. In *GECCO'05: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 169–176, New York, NY, USA, 2005. ACM Press.

[15] R. Rapport. *Nerve Endings: the Discovery of the Synapse*. W. W. Norton & Company, May 2005.

[16] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In *EP'98: Proceedings of the 7th International Conference on Evolutionary Programming*, pages 591–600, London, UK, 1998. Springer-Verlag.

[17] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

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