Particle Swarm and Bayesian Networks Applied to Attribute Selection for Protein Functional Classification

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

Alex A. Freitas
Computing Laboratory and Centre for BioMedical Informatics
University of Kent
Canterbury, CT2 7NF, UK
A.A.Freitas@kent.ac.uk

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

ABSTRACT

The Discrete Particle Swarm (DPSO) algorithm is an optimization method that belongs to the fertile paradigm of Swarm Intelligence. The DPSO was designed for the task of attribute selection and it deals with discrete variables in a straightforward manner. This work extends the DPSO algorithm in two ways. First, we enable the DPSO to select attributes for a Bayesian network algorithm, which is a much more sophisticated algorithm than the Naive Bayes classifier previously used by this algorithm. Second, we apply the DPSO to a challenging protein functional classification data set, involving a large number of classes to be predicted. The performance of the DPSO is compared to the performance of a Binary PSO on the task of selecting attributes in this challenging 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, Data Mining, attribute selection, Naive Bayes classifier, Bayesian networks, bioinformatics.

1. INTRODUCTION

Most of the particle swarm algorithms present in the literature deal only with continuous variables [1, 9, 17]. 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 [24]. Therefore, the design of particle swarm algorithms that deal with discrete variables is pertinent to this field of study.

In [4] we proposed a discrete Particle Swarm Optimization (PSO) algorithm for attribute selection in Data Mining. We will refer to that 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 – it forces the particles to have a constant number of attributes across iterations. The motivation and main innovation of the DPSO algorithm is to interpret the concept of velocity, used in traditional PSO, as “probability"; render velocity as a proportional likelihood and use this information to sample new particle positions. Though the DPSO has been designed for an attribute selection task, it is not limited to this kind of application. With few modifications, the DPSO may potentially be applied to other discrete optimization problems, such as facility location problems [5].

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. Noise or unimportant variables (attributes) may reduce the accuracy and reliability of a classification or prediction model. Unnecessary variables (attributes) also increase the costs of building and running a model – particularly on large data sets. It is therefore important to select an appropriate subset of “good" attributes before performing classification. Attribute selection tries to simplify a data set by reducing its dimensionality and identifying relevant underlying attributes without sacrificing predictive accuracy. As a result, it reduces redundancy in the information provided by the attributes effectively used for prediction. For a more detailed review of the attribute selection task using genetic algorithms see [7].

The DPSO algorithm was designed to 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 [14]. Instead, they represent a combination of selected attributes. In previous work the DPSO was used to select attributes for a Naive Bayes (NB) classifier. The NB classifier was used to predict postsynaptic function in proteins.

This new study extends that previous work in two ways. First, we enable the DPSO to select attributes for a Bayesian network algorithm, which is much more sophisticated than the Naive Bayes algorithm previously used. Second, we apply DPSO to a more challenging protein functional classification data set. This data set has a much larger number of classes to be predicted than the previously

GECCO'07, July 7–11, 2007, London, United Kingdom.

tested postsynaptic data set – which had just two classes to be predicted.

The organization of the paper is: Section 2 briefly addresses Bayesian networks and Naive Bayes classifier. Section 3 shortly discusses PSO algorithms. Section 4 describes the standard Binary PSO algorithm and Section 5 the DPSO algorithm. Section 6 summarizes G protein-coupled receptors (GPCRs). Section 7 reports computational experiments. It also includes a brief discussion of the results obtained. Section 8 presents conclusions and points out future research directions. The following subsection presents notation used throughout this paper.

1.1 Notation

We denote a random variable by an uppercase letter, i.e., X and the state or value of this random variable by a similar lowercase letter, i.e., x. 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 7.1 describes how the measurement $f(\overrightarrow{X}(i))$ is computed in the present work.

2. BAYESIAN NETWORKS AND NAIVE BAYES

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 possible classes. A Naive Bayes classifier assumes that all attributes are conditionally independent of one another [18].

A Bayesian network, by contrast, detects probabilistic dependencies among these attributes and uses this information to benefit the attribute selection process.

A Bayesian network (BN) is a graphical representation of a probability distribution over a set of variables of a given problem domain [10, 20]. This graphical representation is a directed acyclic graph in which nodes represent the variables of the problem and arcs represent conditional probabilistic dependencies among the nodes. The network structure encodes probabilistic dependencies among domain variables and a joint probability distribution quantifies the strength of these dependencies.

An example of a Bayesian network is as follows[1]. Suppose that a doctor is treating a patient who has been suffering from shortness of breath (called dyspnoea). The doctor knows that diseases such as tuberculosis and bronchitis are possible causes for that, as well as lung cancer. The doctor also knows that other relevant information includes whether the patient is a smoker (increasing the chances of cancer and bronchitis) and what sort of air pollution the patient has been exposed to. A positive X-ray would indicate either tuberculosis or lung cancer. The set of variables for this problem and their possible values are shown in Table 1.

Figure 1 shows a Bayesian network representing this problem. For applications of Bayesian networks on evolutionary algorithms and optimization problems see [15, 21].

[1]This is a modified version of the so-called "Asia" problem, [16], given in § 2.5.3.

Table 1: Bayesian network: nodes and values for the lung cancer problem. L = low, H = high, T = true, F = false, Pos = positive and Neg = negative.

Node name	Values
Pollution	{L, H}
Smoker	{T, F}
Cancer	{T, F}
Dyspnoea	{T, F}
X-ray	{Pos, Neg}

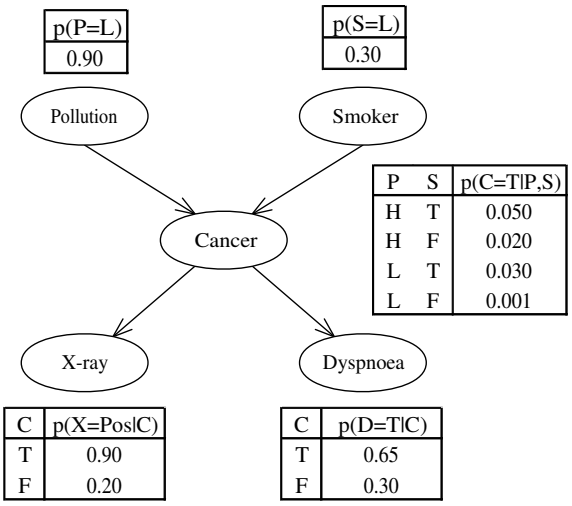

Figure 1: A Bayesian network for the lung cancer problem.

$Parents(X_i)$ represents the set of nodes (attributes) that have a directed edge pointing to X_i. More formally, consider a BN containing ℓ nodes, X_1 to X_ℓ, taken in that order. A particular value of $\overrightarrow{X} = \{X_1, X_2, ..., X_\ell\}$ in the joint probability distribution is represented by:

$$p(\overrightarrow{X}) = p(X_1 = x_1, X_2 = x_2, ..., X_\ell = x_\ell),$$

or more compactly, $p(x_1, x_2, ..., x_\ell)$. The chain rule of probability theory allows us to factorize joint probabilities, therefore:

$$\begin{aligned} p(\overrightarrow{X}) &= p(x_1)\, p(x_2|x_1) \ldots p(x_\ell|x_1, ..., x_{\ell-1}) \\ &= \prod_i p(x_i|x_1, ..., x_{i-1}). \end{aligned} \quad (1)$$

As the structure of a BN implies that the value of a particular node is conditional only on the values of its parent nodes, Equation 1 may be reduced to:

$$p(\overrightarrow{X}) = \prod_i p(X_i|Parents(X_i)). \quad (2)$$

Learning the structure of a BN is an NP-hard problem [2, 3]. Many algorithms developed to this end use a scoring metric and a search procedure. The scoring metric evaluates the goodness-of-fit of a structure to the data. The search procedure generates alternative structures and selects the best one based on the scoring metric. To reduce the search space of networks, only candidate networks in which each node has at most k inward arcs (parents) are considered – k is a parameter determined by the user. In this work we use $\boldsymbol{k = 20}$ to avoid overly complex models.

To generate alternative structures for our BN we used a greedy search algorithm. Starting with an empty network, the greedy search algorithm adds into the network the edge that most increases the score of the resulting network. The search stops when no other edge addition improves the score of the network. Algorithm 1 shows the pseudocode of our generic greedy search algorithm.

Algorithm 1 Pseudocode for a generic greedy search algorithm

Require: Initialize an empty Bayesian network G containing$n$ nodes (i.e., a BN with n nodes but no edges)
1: Evaluate the score of G: $Score(G)$
2: $G' = G$
3: **for** i = 1 to n **do**
4: **for** j = 1 to n **do**
5: **if** $i \neq j$ **then**
6: **if** there is no edge between the nodes i and j in G' **then**
7: Modify G' by adding an edge between the nodes i and j in G' such that i is a parent of j: $(i \rightarrow j)$
8: **if** the resulting G' is a DAG **then**
9: **if** $(Score(G') > Score(G))$ **then**
10: $G = G'$
11: **end if**
12: **end if**
13: **end if**
14: **end if**
15: $G' = G$
16: **end for**
17: **end for**

In this work we evaluate the "goodness-of-fit" (score) of a network structure to the data using an unconventional scoring metric. To evaluate the score of candidate networks we proceed as follows. We divide the data set into 10 equally sized folds. For all class levels each fold maintains roughly the same proportion of classes present in the whole data set before division. This is called stratified cross-validation. Eight of the ten folds are used to compute the probabilities for the bayesian network. The ninth fold is used as validation set and the tenth fold as test set. During the search for the network structure only the validation set is used to compute predictive accuracy. The score of the candidate networks is given by the predictive accuracy of the classification of the proteins in the validation set. The network that shows the highest predictive accuracy on the validation set is then used to compute the predictive accuracy on the test set. Once the network structure is selected, the nine folds are merged and this merged data set is used to compute the probabilities for the selected Bayesian network. The predictive accuracy (reported as the final result) is then computed on the previously untouched test set fold. Every fold will be once used as validation set and once used as test set. This process is discussed again, somewhat in more details, in subsection 7.1 when the computation of a fitness function is presented. A similar process is adopted for the computation of the predictive accuracy using the Naive Bayes classifier.

3. 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 [14]. 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 [13]. 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 [12, 14].

As a whole, 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 particle positions 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 position $\overrightarrow{X}(i)$. 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.

4. THE STANDARD BINARY PSO ALGORITHM

The standard binary version of the PSO algorithm [14] 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 indicates that the respective attribute is not selected. A value of 1 means that it is selected.

4.1 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 $X_{(i,d)} = 0$. We then record this exactly initial population to be used as the initial population by the DPSO algorithm. This is to try to make the comparison between both algorithms as fair as possible.

4.2 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.

4.3 Updating the velocities for the standard Binary PSO algorithm

Every particle $\overrightarrow{X}(i)$ 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:

$$v_{(i,d)} = w\, v_{(i,d)} + \varphi_1(b_{(i,d)} - X_{(i,d)}) + \varphi_2(g_{(d)} - X_{(i,d)}), \qquad (3)$$

where w ($0 < w < 1$), called the inertia weight, is a constant value chosen by the user. Equation 3 is a standard equation used in PSO algorithms to update the velocities [11, 22]. 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).

4.4 Sampling new particle positions for the standard Binary PSO algorithm

New particle positions are sampled as follows. For each particle $\overrightarrow{X}(i)$ 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:

$$X_{(i,d)} = \begin{cases} 1, & \text{if}(\text{rand} < S(v_{(i,d)})) \\ 0, & \text{otherwise;} \end{cases} \qquad (4)$$

where $0 \leq \text{rand} \leq 1$ is a uniform random number and

$$S(v_{(i,d)}) = \frac{1}{1 + \exp(-v_{(i,d)})}$$

is the sigmoid function. Equation 4 is a standard equation used to sample new particle positions in the Binary PSO algorithm [14]. 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.

5. THE DISCRETE PSO ALGORITHM (DPSO)

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.

5.1 Encoding of the particles for the 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}.

5.2 The initial population for the DPSO algorithm

The initial population of solutions used by the DPSO is always identical to the initial population used by the Binary PSO. They differ only in the way in which solutions are represented. We translate all candidate solution in the initial population (of the binary PSO to the Discrete PSO population) in the following way: the index of every attribute that has value 1 is copied to the new solution (particle) of the DPSO initial population. For instance, a solution equal to (1, 0, 1, 1, 0) is translated into {1, 3, 4}.

5.3 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. We use $\dot{V}(i)$ to represent an array of proportional likelihoods and $\dot{v}$ to represent one of its components. 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:

$$\dot{V}(i) = \begin{pmatrix} \text{proportional likelihood row} \\ \text{attribute index row} \end{pmatrix}.$$

Each of the n elements in the first row of $\dot{V}(i)$ represents the proportional likelihood that an attribute be selected. The second row of $\dot{V}(i)$ 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 $\dot{V}(i)$ are set to 1, for example:

$$\dot{V}(i) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}.$$

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 used to update the proportional likelihoods $\dot{v}_{(i,d)}$. These factors determine the strength of the contribution of $\overrightarrow{X}(i)$, $\overrightarrow{B}(i)$ and $\overrightarrow{G}$ to the adjustment of every coordinate $\dot{v}_{(i,d)} \in \dot{V}(i)$. Note that α, β and γ are parameters chosen by the user. The contribution of these parameters to the updating of $\dot{v}_{(i,d)}$ 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, α = 0.10, β = 0.12, γ = 0.14, $\overrightarrow{X}(i)$ = {2, 3, 4}, $\overrightarrow{B}(i)$ = {3, 5, 2}, $\overrightarrow{G}$ = {5, 2} and also:

$\dot{V}(i) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$, the updated $\dot{V}(i)$ would be:

$$\dot{V}(i) = \begin{pmatrix} 1 & 1+\alpha+\beta+\gamma & 1+\alpha+\beta & 1+\alpha & 1+\beta+\gamma \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}.$$

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 $\dot{V}(i)$ 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.

5.4 Sampling new particle positions for the DPSO algorithm

The proportional likelihood array $\dot{V}(i)$ 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 $\dot{V}(i)$ 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

$$\dot{V}(i) = \begin{pmatrix} 1 & 1.36 & 1.22 & 1.1 & 1.26 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}.$$

The multiplied proportional likelihood array would be:

$$\dot{V}(i) = \begin{pmatrix} 1 \times \varphi_1 & 1.36 \times \varphi_2 & 1.22 \cdot \varphi_3 & 1.1 \cdot \varphi_4 & 1.26 \cdot \varphi_5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix},$$

where φ_1, ..., φ_5 are uniform random numbers independently drawn on the interval (0, 1). Suppose that the multiplied array $\dot{V}(i)$ looks like this:

$$\dot{V}(i) = \begin{pmatrix} 0.11 & 0.86 & 0.57 & 0.62 & 1.09 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}.$$

The new particle position is then defined by ranking the columns in $\dot{V}(i)$ 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 $\dot{V}(i)$) follow their respective proportional likelihoods. For example, ranking the array:

$$\dot{V}(i) = \begin{pmatrix} 0.11 & 0.86 & 0.57 & 0.62 & 1.09 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix},$$

we would obtain $\dot{V}(i) = \begin{pmatrix} 1.09 & 0.86 & 0.62 & 0.57 & 0.11 \\ 5 & 2 & 4 & 3 & 1 \end{pmatrix}$.

After ranking the array $\dot{V}(i)$, the first k indices (in the second row of $\dot{V}(i)$), from left to right, are selected to compose the new particle position. The constant k represents the length of the particle $\overrightarrow{X}(i)$, the particle associated to the ranked array $\dot{V}(i)$. Thus, if particle $\overrightarrow{X}(i)$, a particle associated to the multiplied and sorted array:

$$\dot{V}(i) = \begin{pmatrix} 1.09 & 0.86 & 0.62 & 0.57 & 0.11 \\ 5 & 2 & 4 & 3 & 1 \end{pmatrix},$$

has length 3, the first 3 indices from the second row of $\dot{V}(i)$ would be selected to compose the new particle position. Based on the array $\dot{V}(i)$ 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 position, 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 4.2.

6. G PROTEIN-COUPLED RECEPTORS (GPCRS)

G protein-coupled receptors (GPCRs) are a protein family of transmembrane receptors. Their function is to transduce signals that induce a cellular response to the environment. GPCRs are the largest protein family known and they are involved in all types of stimulus-response pathways, from intercellular communication to physiological senses. GPCRs are of much interest to the pharmaceutical industry for these proteins are involved in many pathological conditions, which led to GPCRs being the target of 40% to 50% of modern medicinal drugs [6].

In this work we use the GPCR-PROSITE data set of proteins previously used in [8]. The data set contains 190 proteins. The proteins are represented by a set of 127 PROSITE patterns. 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. The proteins in this data set are grouped into families and subfamilies in a hierarchical fashion. There are three levels of hierarchy. The first level has 8 classes (families), the second and third levels have 32 classes (subfamilies) each one (some proteins are classified only up to the second hierarchical level and have no class at the third level). The objective of our algorithms is to classify each protein into its most suitable family in each level. In this work the classification of the proteins is performed for each class level individually. For instance, given protein X a conventional "flat" classification algorithm assigns X's class at the first class level only. Once protein X has been classified at the first class level, the conventional flat classification algorithm is again applied to assign a class to protein X at the second level – no information about X's class at the previous level is used. The same process is used to assign a class to protein X at the third class level.

7. EXPERIMENTS

In this section, we report and discuss computational experiments. The quality of a candidate solution (fitness) is evaluated in three different ways: (1) by a baseline algorithm (using all possible attributes); (2) by the Binary PSO; and (3) by the Discrete PSO (DPSO). Each of these algorithms computes the fitness of every given solution using two distinct techniques: (a) using a Naive Bayes classifier; and (b) using a Bayesian network. For the Binary PSO and DPSO 30 independent runs are performed for each single fold. The results obtained, averaged over 30 runs, are reported in Table 2.

7.1 Experimental Methodology

The fitness function $f(\overrightarrow{X}(i))$ of any particle $\overrightarrow{X}(i)$ is computed as follows. $f(\overrightarrow{X}(i))$ is equal to the predictive accuracy achieved by the Naive Bayes classifier (and the Bayesian network) on the GPCR-PROSITE 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 (for each class level) with an acceptable accuracy. We define the accuracy as acceptable if it is equal to or better than the accuracy obtained by the classifica-

Table 2: Results for the GPCR-PROSITE data set.

127 ATTRIBUTES		AVERAGE PREDICTIVE ACCURACY			AVERAGE NUMBER OF SELECTED ATTRIBUTES	
METHOD	CLASS LEVEL	USING ALL ATTRIBUTES	BINARY PSO	DISCRETE PSO	BINARY PSO	DISCRETE PSO
NAIVE BAYES	1	71.27±2.08	72.88±2.40	*73.05±2.31	85.60±2.84	*74.90±3.48
	2	30.00±2.10	31.34±2.47	*32.60±2.31	101.50±3.14	*83.80±4.64
	3	20.47±0.96	21.47±1.16	*23.25±1.08	102.30±3.77	*87.50±4.25
BAYESIAN NETWORK	1	78.05±2.33	79.03±2.57	*80.54±2.46	78.50±3.50	*65.50±3.41
	2	39.08±2.67	40.31±2.85	*43.24±4.67	94.10±3.70	*73.30±2.67
	3	24.70±1.83	26.14±2.11	*28.97±2.77	94.90±3.90	*77.60±4.35

The best result on each line for each performance criterion is marked with an asterisk (*).

tion performed considering all the 127 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 GPCR-PROSITE 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 or Bayesian network. For more information on wrapper and other attribute selection approaches see [25].

The computational experiments involved a 10-fold cross-validation method [25]. First, the 190 records in the GPCR-PROSITE data set were divided into 10 equally sized folds. The folds were randomly generated but under the following criterion. The proportion of classes in every single fold must be similar to the one found in the original data set containing all the 190 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 and the Bayesian network use the remaining 8 folds to compute the probabilities required to classify new examples. Once those probabilities have been computed, the Naive Bayes classifier (NB) and the Bayesian network (BN) classify the examples in the validation set. The accuracy of this classification on the validation set is the value of the fitness functions $f_{\text{NB}}(\overrightarrow{X}(i))$ and $f_{\text{BN}}(\overrightarrow{X}(i))$. After the run of the PSO algorithm is completed, the 9 folds are merged into a full training set. The Naive Bayes classifier and the Bayesian network are 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 fold), which was never accessed during the run of the algorithms. 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 190 original attributes**: all possible attributes are used by the Naive Bayes classifier and the Bayesian network.

(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 and the Bayesian network.

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

Since the Naive Bayes and Bayesian network classifiers that we used are deterministic, only one run (for each of these algorithms) is performed for the classification using all the 127 attributes. For the Binary PSO and the DPSO algorithms 30 independent are performed for each fold. Results reported are averaged over these 30 independent runs. The population size used for both algorithms (Binary PSO and DPSO) 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 [23]. Other choices of parameter values for the DPSO 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:

$$\textit{Standard accuracy rate} = \frac{TP + TN}{TP + FP + FN + TN} \tag{5}$$

is used to assess the accuracy rate of a classifier (where TP, TN, FP, FN are the numbers of true positives, true negatives, false positives and false negatives, respectively [25]). Nevertheless, if the class distribution is highly unbalanced, Equation 5 is an ineffective way of measuring the accuracy rate of a model. For instance, in many problems it is easy to maximize Equation 5 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 [19]. This measurement is given by the equation:

$$\textit{Predictive accuracy rate} = TPR \cdot TNR\,, \tag{6}$$

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

Note that if any of the quantities TPR or TNR is zero, the value returned by Equation 6 is also zero.

7.2 Discussion

Results are reported in Table 2. First, we discuss the results obtained by the three algorithms using the Naive Bayes classifier. To assess the performance of the algorithms we consider two criteria: (1) maximizing predictive accuracy; and (2) finding the smallest

subset of attributes. Comparing the first criterion, accuracy, we note that both versions of the PSO algorithm did better (in all class levels) than the baseline algorithm using all attributes. Furthermore, the DPSO algorithm did slightly better than the Binary PSO algorithm in all class levels. Nevertheless, the difference in the predictive accuracy performance between these algorithms is, in some cases, not statistically significant. Table 3 shows the results of a paired two-tailed *t*-test for the predictive accuracy of the Binary PSO versus the predictive accuracy of the DPSO (at a significance level of 0.05).

Table 3: Binary PSO vs. DPSO (ACCURACY) : paired two-tailed *t*-test for the predictive accuracy (significance level 0.05).

CLASS LEVEL	Naive Bayes	Bayesian network
1	t(9) = 0.467, p = 0.651	t(9) = 3.407, p = 0.007
2	t(9) = 2.221, p = 0.053	t(9) = 3.200, p = 0.010
3	t(9) = 3.307, p = 0.009	t(9) = 3.556, p = 0.006

According to Table 3, using Naive Bayes as classifier the only statistically significant difference in performance (in terms of predictive accuracy) between the algorithms (Binary PSO and DPSO) is at the third class level. By contrast, using Bayesian networks as classifier the difference in performance is statistically significant at all class levels.

However, the discriminating factor between the performance of these algorithms is on the second comparison criterion – finding the smallest subset of attributes. The DPSO not only outperformed the binary PSO in predictive accuracy, but also did so using a smaller subset of attributes in all class levels. Moreover, when it comes to effectively pruning the set of attributes, the difference in performance between the Binary PSO and the DPSO is always statistically significant. Table 4 shows that.

Table 4: Binary PSO vs. DPSO (ATTRIBUTES) : paired two-tailed *t*-test for the number of attributes selected (significance level 0.05).

CLASS LEVEL	Naive Bayes	Bayesian network
1	t(9) = 7.248, p = 4.8E-5	t(9) = 8.2770, p = 1.6E-5
2	t(9) = 9.052, p = 8.1E-6	t(9) = 14.890, p = 1.2E-7
3	t(9) = 6.887, p = 7.1E-5	t(9) = 9.1730, p = 7.3E-6

Second, we discuss the results obtained using the Bayesian network algorithm as a classifier. Again, the predictive accuracy attained by both versions of the PSO algorithm surpassed the predictive accuracy obtained by the baseline algorithm in all class levels. DPSO obtained the best predictive accuracy of all algorithms in all three class levels. In terms of the second comparison criterion, finding the smallest subset of attributes, again DPSO always selected the smallest subset of attributes in all hierarchical levels.

Comparing the performance of the classifiers (Naive Bayes vs. Bayesian networks), we note that Bayesian networks did a much better job. For all three class levels the predictive accuracy obtained by the algorithms (baseline, Binary PSO and DPSO) using Bayesian networks was significantly better than the predictive accuracy obtained using Naive Bayes classifier. The Bayesian networks also enabled the two PSO algorithms to do the job using fewer selected attributes.

The results emphasize the importance of taking correlations among attributes into account when doing attribute selection. When these correlations are ignored, predictive accuracy is adversely affected.

8. CONCLUSIONS

Computational results show that the use of unimportant attributes tend to derail classifiers and hurt classification accuracy. Using fewer attributes, the Binary PSO and the DPSO algorithms obtained better predictive accuracy (in 100% of the cases) than the classification performed using all possible attributes. Previous work had already shown that the DPSO algorithm performs better than the Binary PSO in the task of attribute selection [4]. Even if the improvement in predictive accuracy is not significant, by selecting fewer attributes the DPSO certainly enhance computational efficiency of the classifier.

The original work, however, questioned whether the difference in performance between these two algorithms was attributable to variations in the initial population of solutions. To overcome this possible advantage/disadvantage for one algorithm or the other, the present work used the same initialization for both algorithms. Computational results show that, even using the same initial conditions, the DPSO is still outperforming the Binary PSO in both predictive accuracy and number of selected attributes. The DPSO is arguably not too different from traditional PSO but still the algorithm has some features that enable it to improve over binary PSO.

Another interesting result from the experiments is the clear difference in performance between Naive Bayes and Bayesian networks used as classifiers. Bayesian networks outperformed Naive Bayes classifier in all experiments and in all hierarchical class levels.

The hierarchical classification performed in this work was a flat classification. The algorithms did not use the information of the class assigned to an example (protein) in one level to help the prediction of the class of at the next hierarchical level. In future work we intend to develop an algorithm that takes advantage of this information.

9. ACKNOWLEDGMENTS

Thanks to Nick Holden for kindly providing us with the biological data sets used in this work. The authors would also like to thank EPSRC (grant *Extended Particle Swarms* GR/T11265/01) for financial support.

10. 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] R. R. Bouckaert. Properties of Bayesian belief network learning algorithms. In I. R. L. de Mantaras and e. D. Poole, editors, *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 102–109, Seattle, WA, USA, 1994. Morgan Kaufmann.

[3] D. M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research, November 1994.

[4] E. S. Correa, A. A. Freitas, and C. G. Johnson. A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set. In M. K. et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-2006*, pages 35–42, Seattle, WA, USA, July 2006. ACM Press.

[5] 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.

[6] D. Filmore. It's a GPCR world. *Modern drug discovery*, 11(7):24–28, November 2004.

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

[8] N. Holden and A. A. Freitas. Hierarchical classification of g-protein-coupled receptors with a pso/aco algorithm. In *Proc. IEEE Swarm Intelligence Symposium (SIS-06)*, pages 77–84. IEEE Press, June 2006.

[9] 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.

[10] F. V. Jensen. *Bayesian networks and decision graphs*. Springer-Verlag, 1st edition, July 2001.

[11] 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.

[12] 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.

[13] 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.

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

[15] P. Larrañaga, R. Etxeberria, J. A. Lozano, B. Sierra, I. naki Inza, and J. M. Peña. A review of the cooperation between evolutionary computation and probabilistic models. In *Second Symposium on Artificial Intelligence - CIMAF-1999*, pages 314–324, Havana, Cuba, March 1999. Special Session on Distributions and Evolutionary Computation.

[16] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistics Society 50*, 2:157–224, 1988.

[17] 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.

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

[19] 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.

[20] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1st edition, September 1988.

[21] J. M. Peña, J. A. Lozano, and P. Larrañaga. Globally multimodal problem optimization via an estimation of distribution algorithm based on unsupervised learning of bayesian networks. In *Evolutionary Computation*, volume 13, pages 43–66. MIT Press, January 2005.

[22] 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.

[23] 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.

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

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