2005 IEEE Swarm Intelligence Symposium, 8-10 June, Pasadena, California, USA

A HYBRID PARTICLE SWARM/ANT COLONY ALGORITHM FOR THE CLASSIFICATION OF HIERARCHICAL BIOLOGICAL DATA

Nicholas Holden

Computing Laboratory, University of Kent
Canterbury, CT2 7NF, UK
nh56@kent.ac.uk

Alex A. Freitas

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

ABSTRACT

This paper proposes a hybrid PSO/ACO algorithm for hierarchical classification, where the classes to be predicted are arranged in a tree-like hierarchy. The performance of the algorithm is evaluated on a challenging biological data set, involving the hierarchical functional classification of enzymes. The proposed algorithm is compared with an existing PSO for classification, which was also adapted for hierarchical classification.

1. INTRODUCTION

The discovery of a new protein and its function was once deemed worthy of a paper in its own right. Now with the automation of the processes involved with the discovery of new proteins it is almost viewed as common place. Due to the large amount of new proteins being discovered, automated processes are also needed to find what purpose a protein might have within a biological system. This paper deals with the hierarchical functional classification of enzymes (a sub set of proteins). Enzymes are nature's catalysts and usually are more effective at catalysing reactions than their non-biological counterparts. An example of such effective enzymes are those found in biological washing powder.

We propose a new hybrid PSO (Particle Swarm Optimization)/ACO (Ant Colony Optimization) classification algorithm tailored to this challenge. More specifically to cope with the extremely large number of attributes and classes, and the categorical (nominal, non-numerical) data often associated with the problem. We hope this paper will serve as an introduction into the complex and challenging realm of hierarchical classification of biological data sets. In any case, it should be noted that the proposed hybrid PSO/ACO algorithm is generic enough to be applied to other challenging classification problems.

The basic motivation for designing the hybrid algorithm was to make PSO more effective in coping with categorical attributes using the pheromone-based mechanism of ACO, as will be discussed later. The proposed hybrid PSO/ACO algorithm is compared with an existing PSO for classification, which was also adapted for hierarchical classification. Both algorithms follow the top-down approach for hierarchical classification, using the predictions of higher-level classes to guide the search for rules predicting lower-level classes.

2. PROTEINS AND ENZYMES

Proteins are the active building blocks of all life and carry out most of the functions involved with it (there are always exceptions in biology). Enzymes are a sub set of proteins; they are catalysts which are used to speed up and make possible many of the chemical reactions that take part within the cell, without being altered themselves. Enzymes are assigned EC codes (enzyme commission numbers), which are 4 digit numbers that represent the type of chemical reaction the enzyme in question catalyses [6]. Each digit corresponds to a level in the hierarchy. For instance, EC 3.1.4.1 is an enzyme with class value 3 in the first level, class value 1 in the second level, etc.

Proteins are formed from a number of amino acids chained together. There are 20 different amino acids that occur naturally, and a linear sequence of these amino acids is known as the primary structure. The secondary structures seen in proteins are the 3D shapes that form locally in each protein and may be repeated throughout it. They also may be common to multiple proteins. These common patterns and domains include helixes, sheets, active sites which catalyse reactions in enzymes (a sub set of proteins), various sites which allow functions of a protein to be turned on and off etc. From a data mining point of view these regions are very interesting as they work together to produce the behaviour observed in proteins and so must produce patterns that can be analysed. A number of databases of these common structures have been created, including the Prosite database [7], which is used in this work. This database contains unique "fingerprint" style entries which are designed to be used to identify the function of unknown proteins.

The tertiary structure can be described as the overall shape formed when the chemically attracted portions cause the protein to fold. These individual attractions are quite weak, but because there are so many of them the resulting protein can be structurally very strong, although when heat is applied they tend to unfold or denature. (This happens, e.g.,

when eggs are cooked). The quaternary structure is where proteins join together to form more complicated structures, such as cell walls or spiders silk.

3. HIERARCHICAL CLASSIFICATION

Data mining consists of a set of concepts and techniques used to find useful patterns within a set of data [4], [5]. In this project the discovered knowledge is represented as classification rules. A rule consists of an antecedent (a set of attribute values) and a consequent (class):

IF <attrib = value> AND ... AND <attrib = value>
THEN <class>

The consequent of the rule is the class predicted by the rule for the records (examples) where the predictor attributes hold. An example rule might be IF <Salary = high> AND <Mortgage = No> THEN <Good Credit>. This kind of knowledge representation has the advantage of being intuitively comprehensible to the user. This is important, because the general goal of data mining is to discover knowledge that is not only accurate, but also comprehensible [4][5].

In this project the classes are arranged in a tree structure where each node (class) has only one parent. Hierarchical class datasets present new challenges when compared to flat class datasets. The main challenge comes from the extra complexity associated with such datasets, the fact that many (depending on the depth) more classes must be assigned to the examples, and the prediction of a class becomes increasingly difficult as deeper levels are considered, due to the smaller number of examples per class. Note that although the functional classification of enzymes is an important bioinformatics problem, previous applications of data mining to this problem typically ignore the class hierarchy, focusing on predicting classes in just one level of that hierarchy [9][10].

The simplest way to deal with hierarchical classification is to ignore the hierarchy completely and so only predict classes at the bottom most level, indirectly predicting the classes at higher levels. This approach avoids the aforementioned extra complexity at the expense of not discovering simpler knowledge expressed by higher level rules (based on a larger number of examples). It discovers only lowest level rules based on a small number of examples, and so those rules tend to be inaccurate.

The second approach is to again ignore the fact that the classes are in a hierarchy at the training stage and classify each node separately. It can be argued that this will produce the most complete set of rules, and so if this is taken into account at classification (on the test set) there is a higher chance of making an accurate prediction. However the knowledge found is overly complex and so harder to comprehend, also this approach is more computationally expensive.

The third approach (which is used in this project) uses the divide and conquer principle [1]. If class 1.X.X.X (where X denotes any digit) is predicted at the first level and node 1 has only the child nodes 1.1.X.X and 1.2.X.X, only these two nodes should be considered and not the children belonging to node 2.X.X.X. This holds both during training and test set classification. It does however create problems of misclassification; if an example is misclassified at a higher node then it has no chance of being correctly classified at lower nodes. However, it has the advantages of producing a more complete set of rules than the first approach without it being needlessly (as with the second approach) complex, while also using the nature of the hierarchical structure of the data to optimise performance.

4. THE HYBRID PSO/ACO ALGORITHM

Although PSO and ACO algorithms have been developed for the classification task we do not feel that either of the current algorithms is ideally suited for this problem, for the following reasons stated.

Although the ACO classification algorithm has already been shown [12] at least competitive with the industrial standard C5.0 algorithm, the unusually large amount of attributes and classes associated with this problem mean an *extremely* large amount of computation time is required. This is because a computationally expensive rule pruning procedure is required at every inner iteration of the standard ACO algorithm. Also for the algorithm to work with continuous attributes they have to be previously discretised, which can decrease classification accuracy and increase computational time at this pre-processing step.

A PSO algorithm has also been developed for classification [11], however we believe it can be improved by hybridising it with ACO. The classification task usually involves a mixing of both continuous and categorical (nominal, non-numerical) attribute values. Although a “standard” binary/discrete PSO algorithm exists [14], it does not deal with categorical values in a natural fashion when compared to ACO. In particular, the standard PSO for coping with binary attributes represents a particle by a bitstring, where each binary value such as true or false is encoded as 1 or 0. The usual notion of “velocity” (a core concept in a PSO for coping with continuous variables) is replaced by the notion of “predisposition of taking the value 1 (rather than 0)”. This approach was designed to cope with binary attributes/decisions, but not explicitly designed to cope with

multi-valued (i.e, having more than two values) categorical attributes.

Sousa et al. extended the standard binary PSO to cope with multi-valued categorical attributes [11], developing a Discrete PSO (DPSO) algorithm for discovering classification rules. In essence, in DPSO each value for a categorical attribute is assigned an index number. This number is then converted into a binary string. An extra bit is also added for each attribute to decide if that attribute is to be included in the resulting rule. Once a categorical attribute has been converted into a binary string, the standard binary PSO can then be applied.

However, this encoding approach introduces some problems. In particular, due to the conversion to binary, the bits of the string to be optimised by the algorithm interact with each other to form the index number. This interaction is adds an extra layer of complexity and confusion for the algorithm, because PSO works by trying to find the optimal value of each bit individually. I.e., each bit is a "dimension" from the point of view of PSO, and the "velocity" of a bit – its propensity of taking the value 1 – is computed independent from other bits that are part of the encoding of the same attribute, ignoring important bit interactions.

Another (related) problem is that the numerical index assigned to a categorical value and the subsequent binary encoding scheme will affect the result of any particle interaction. For instance, consider the categorical attribute *Marital Status* with the following four nominal values: *single*, *married*, *divorced*, *widowed*. There are 24 different ways of mapping these nominal values into numerical indices in the range 0 to 3 and subsequent convert them into two binary digits. (Each permutation of the nominal values corresponds to a different mapping, and the number of permutations is 4! = 24.) The choice of a mapping is arbitrary, but it affects particle interactions. For instance, *single* and *widowed* might be converted to 00 and 11, in which case the two nominal values would be totally different in their binary representation, or they might be converted to 00 and 01, in which case they would be just partially different in their binary representation. This affects the computation of the "velocities" of individual bits, since those velocities depend on the differences between bit values of the current particle and bit values in the particle's best past position and best neighbour. As there is no ordering in categorical data, ideally an encoding scheme should be chosen where all bit strings are equally pairwise similar (i.e., every pair of bit strings has the same Hamming distance), which is not possible in general, as shown in the above example.

Another major problem with the current DPSO classification algorithm, in the context of large and complex data sets, is that the population is initialised randomly. This is acceptable with a low number of attributes, as there is a large probability of randomly producing a particle with a non-zero fitness. However it becomes a major problem in classification problems when large numbers of attributes are present, as many or all of the particles may have zero finesses in the population. The rules that particles represent may also be needlessly long, increasing fitness evaluation time and decreasing the comprehensibility of the resulting rule. If all or most of the particles have zero finesses then the convergence time will be significantly increased while the particles randomly search to find a position with a non-zero fitness, or they may converge to a bad position simply because there are only a few non-zero positions "known" in the population.

The hybrid algorithm addresses these issues by combining characteristics of PSO and ACO algorithms. ACO has been shown good at solving classification problems with categorical data [13], as it does not introduce any artificial ordering among attribute values and features none of the encoding problems of DPSO previously mentioned. PSO has been shown good at solving optimisation problems with continuous values, which are often present in data mining. The original ACO algorithm for classification, Ant-Miner [12], requires that every rule be pruned right after the rule is created. This is due to the way the rules are constructed (incrementally, one-condition-at-a-time) and the way the problem is represented in terms of pheromone. Ant-Miner's rule pruning tends to be effective in improving a rule, but it is very computationally expensive when the rule has many attributes, which is a serious limitation in the context of the data set mined in this project. The hybrid algorithm does not require such pruning because of the addition of an indifference entry in the pheromone matrices (pruning is still carried out on the final best rule generated by the population) and the fact that a separate matrix is used for each attribute. The overall effect is to produce a "swarm of ant colonies".

4.1 SEQUENTIAL RULE DISCOVERY AND PARTICLE REPRESENTATION

The algorithm uses a sequential covering approach [4] to discover one-classification-rule-at-a-time, as shown in Pseudocode 1. It starts by initialising the rule set (RS) with the empty set. Then, for each hierarchical class level and for each of the classes to be predicted, the algorithm performs a WHILE loop. Each iteration of this loop performs one run of the PSO/ACO algorithm, returning the best discovered rule predicting examples of the current class (C). This rule is added to the rule set, and the examples correctly covered by that rule are removed from the sub training set (TS). An example is said to be correctly covered by a rule if that

example satisfies all the terms (attribute-value pairs) in the rule antecedent ("IF part") and it has the class predicted by the rule. This WHILE loop is performed as long as the number of uncovered examples of the class C is greater than a user-defined threshold, the maximum number of uncovered examples per class (MaxUncovExampPerClass). After discovering rules for all classes at all levels, the algorithm returns RS, the discovered rule set. To apply this algorithm to the hierarchical classification problem the training set TS is a sub set of the entire training set available. TS contains all the examples from classes with the same parent class as C. Class C is the "positive class" and its sibling classes are the "negative classes". So if classes 1.1.X.X, 1.2.X.X and 2.1.X.X exist and class C is 1.1.X.X, then examples from 1.1.X.X and 1.2.X.X would be used as positive and negative examples in TS, respectively, whereas examples from class 2.1.X.X would be ignored. At the top most level the entire training set is used as TS. The training examples that have been removed right after the discovery of a rule are replaced in TS when the algorithm starts creating rules for a new class.

```
RS = ∅   /* initially, Rule Set is empty */
FOR EACH LEVEL L
   FOR EACH class C at L
      TS = {all training examples belonging to classes at
            level L with the same parent as C}
      WHILE (number of uncovered training examples
             of class C > MaxUncovExampPerClass)
         Run the PSO/ACO algorithm to discover the
         best rule predicting class C, called BestRule
         RS = RS ∪ BestRule
         TS = TS – {training examples correctly
                    covered by discovered rule}
      END WHILE
   END FOR
END FOR
```

Pseudocode 1: Sequential covering approach used by the hybrid PSO/ACO algorithm

Each particle represents the antecedent of a candidate classification rule. The rule's class is fixed for all the particles in each run of the algorithm, since – as shown in Pseudocode 1 – each run of the algorithm aims at discovering the best rule for a fixed class. This approach has the advantage of avoiding the problem of having different particles predicting different classes in the same population, which would hinder the effective exploitation of the PSO principle of "imitating the best neighbour".

Although not dealt with in this paper the algorithm can easily be extended to cope with continuous (real-valued) attributes. Standard particle swarm optimisation is particularly well suited to this problem and so a continuous value can be directly represented as a component of the vector associated with a particle and processed using the standard formulas for PSO [14]. A simple approach would be to define upper and lower bounds for the continuous attribute in the rule, an example might be IF 21 ≤ age ≤ 35 THEN wage = medium.

On the other hand, for the reasons discussed earlier, categorical (nominal) attributes are handled in a special way, as follows. A particle contains a number of pheromone matrices equal to number of categorical attributes in the data set. Each pheromone matrix contains values for pheromones for each possible value that that attribute can take [12] plus a flag value (the indifference flag) indicating whether or not the attribute (ProSite pattern) is selected to occur in the decoded rule. The particle representation for categorical attributes is shown in graphical form in Figure 1, where each attribute value and the indifference flag are represented as slots in a roulette wheel. This analogy is appropriate for explaining the process of moving the particles with respect to categorical attributes, as discussed in the next section.

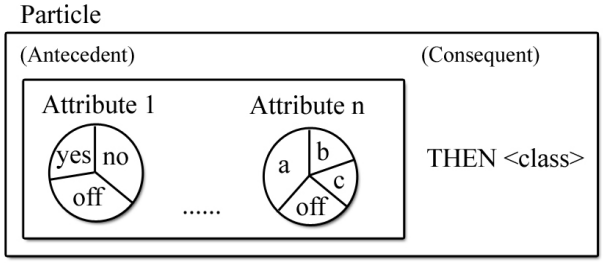

Figure 1: Particle representation considering categorical attributes only

4.2 MOVING THE PARTICLE WITH RESPECT TO CATEGORICAL (NOMINAL, NON-NUMERIC) ATTRIBUTES

At each iteration, each categorical attribute in the rule antecedent represented by each particle has its value chosen, in order to give a particle a fixed position and so quality. This is the decoding process. An attribute value is chosen with probability proportional to its pheromone value. This fixed position and so quality is used to update the particle's pheromone matrices in the next iteration. If the new position has a higher quality than any position the particle has ever occupied then it is set as the particle's past best position. To update the values in the pheromone matrices of the current particle, the past best, current and its best neighbour's positions are used. The quality of these three positions, multiplied by individual random learning factors as usual in PSO, are added to the values in the appropriate entries in the pheromone matrices of the current particle. For instance, suppose that one of these positions (corresponding to a decoded classification rule), say the rule decoded from the particle's best neighbour, does not contain any values for a

given attribute, i.e., the decoded rule has indifference flag set to off in that given attribute. Then the pheromone value associated with the indifference flag (the "off" slot in Figure 1) of that attribute of the current particle will be updated by adding, to the current pheromone value, an amount which is equal to the quality of the best neighbour rule multiplied by a random number. As another general example, suppose the rule from the particle's best past position includes a value equal to *yes* for the *i*-th attribute. Then the pheromone value associated with the value *yes* of the *i*-th attribute of the current particle will be updated by adding, to the current pheromone value, the product of the best past position rule times a random number. After the qualities of the three decoded rules (from best past position, current position, and best neighbour position) have been added to the corresponding pheromone values of the current position), the pheromone matrix is normalised to simulate evaporation on unused attribute values, in the same style of the pheromone evaporation procedure used in [12]. More precisely, the formulas for updating a particle's pheromone (the probabilities of choosing attribute values) are as follows:

$$(1)\ \tau_{cij} = \tau_{cij} + (\varphi_1 * Q_c), \forall ij \in \text{CurrentRule}$$

$$(2)\ \tau_{cij} = \tau_{cij} + (\varphi_2 * Q_p), \forall ij \in \text{BestPastRule}$$

$$(3)\ \tau_{cij} = \tau_{cij} + (\varphi_3 * Q_l), \forall ij \in \text{BestLocalRule}$$

$$(4)\ \tau_{cij} = \frac{\tau_{cij}}{\sum_{j=1}^{a_i+1} \tau_{cij}}$$

Where τ_{cij} is the amount of pheromone in the current particle *c*, for attribute *i*, for value *j*. Q_c is the quality of the rule represented by the *current* position of the current particle, Q_p is the quality for the rule represented by the best *past* position of the current particle and Q_l is the quality of the rule represented by the best *local* neighbour's position. φ is a random learning factor in the range 0..1. We stress that in Equations (2) and (3), for each attribute *i*, its value *j* belonging to the best past or best neighbour rule can be different from the value of attribute *i* in the current rule. For instance, in the best local neighbour rule the attribute *i* could have, say, the value *j = yes*, whereas in the current rule the attribute *i* could have the value *no*. In this case, a fraction of the quality of the best local neighbour rule (i.e., $\varphi_3 * Q_l$) would be added to the pheromone entry for *j = yes* (rather than *j = no*) in the current rule. This increase of pheromone for value *yes* of attribute *i* would be increasing the probability that the value *yes* will be chosen the next time that the particle is decoded into a rule, i.e., in the next iteration of the algorithm. Therefore, the mechanism of increasing the pheromone of a given attribute value in the hybrid PSO/ACO corresponds to the mechanism of moving a particle towards that attribute value in conventional PSO. It is also important to bear in mind that in Equations (1)–(3) the index *j* can refer to any of the values of the attribute *i*, including the indifference flag "off" – see Figure 1. Equation (4) normalises the pheromone matrices. In that equation a_i denotes the number of values belonging to the domain of attribute *i*, and the summation is over $a_i + 1$ (rather than just a_i) terms in order to consider the "off" state.

Note that Equations (1)–(3) are quite different from the particle movement equations from conventional PSOs in two important ways. First, in conventional PSOs the measure of particle quality (goodness) is used to determine the best past position of a particle and its best neighbour, but the actual value of particle quality is not used to compute velocity. By contrast, in Equations (1)–(3) the actual values of particle quality (Q_c, Q_p, Q_l) are directly used to update the position of the current particle. This characteristic was inherited from ACO principles. Second, in conventional PSO the movement of the current particle is attracted by its best past position and its best neighbour's position. By contrast, in the hybrid PSO/ACO the movement of the current particle is attracted not only by those two positions – Equations (2),(3), but also by the current particle's own position – Equation (1). There is no counterpart to Equation (1) in conventional PSO, but this formula is justified in the hybrid PSO/ACO because the formulas for particle movement are based not only on particle positions, but also on particle qualities. The higher the quality of the current particle, the higher the increase in the amount of pheromone associated with the attribute values belonging to the current rule, and so the larger the tendency for the particle to stay in its current position.

4.3 PARTICLE FITNESS (RULE QUALITY)

The fitness of a given particle is based on the rule it represents, and is given by the following measure of predictive accuracy [12]: Rule Quality = Sensitivity * Specificity, where Sensitivity = TP / (TP + FN) and Specificity = TN (TN + FP), where:

TP (True Positives) is the number of cases that match the rule antecedent (attribute values) and also match the rule consequent (class). These are desirable correct predictions.
FP (False Positives) is the number of cases that match the rule antecedent but do not match the rule consequent. These are undesirable incorrect predictions.
FN (False Negatives) is the number of cases that do not match the rule antecedent but do match the rule consequent. These are undesirable uncovered cases and are caused by an overly specific rule.
TN (True Negatives) is the number of cases that do not match the rule antecedent and do not match the rule

consequent. These are desirable and are caused by a rule's antecedent being specific to its consequent class.

4.4 SEEDING AND PRUNING

If a rule was initialised at random, it might have a quality of 0, if the rule does not cover any example of its predicted class. Our rule initialisation procedure avoids that. The population is initialised in positions with non-zero qualities, and so this is each particle's respective initial past best position. This is achieved by taking an example (record) from the class to be predicted and using its terms (attribute values) as the rule antecedent. This creates an extremely specific rule, covering only the "seed" example. Then to improve the initial quality of the particle, a pruning procedure is applied, as described below. This reduces the time it takes the population to converge and tends to make it converge to a rule of higher quality. We found this procedure is very important when dealing with large numbers of attributes.

Rule pruning simplifies and generalises the rule. It generalises the rule by removing the most irrelevant terms and so increasing the number of examples covered by the rule. It simplifies the rule by removing terms which make the rule overly specific, or which do not affect the quality of the rule. We experimented with two kinds of rule pruning procedures for seeding purposes. In both procedures, the first step is to compute the "quality" of each term (attribute-value pair) occurring in the current rule. As a measure of term quality we use the same formula "Sensitivity * Specificity" explained in section 4.3, with the difference that now, when computing the values of TP, FP, FN and TN for a given term, we try to match each training example with just that term, rather than matching the example with all the terms in the entire rule antecedent. The positive class is, of course, the class predicted by the rule. Hence, term quality is a value in the range 0..1. The larger the value the better the quality of the term. We emphasize that term quality is computed for one-term-at-a-time, ignoring term interactions in the rule, and so it is not a perfect estimator of the term's true predictive power in the rule. However, the computation of this term quality measure is quite fast, so that this rule pruning procedure is much faster than the rule pruning procedure of Ant-Miner, the ACO for discovering classification rules described in [12].

Once the quality of each term in the rule has been computed, one of the following two alternative rule pruning procedures was tried:
1) Selecting a fixed number *K* of relevant terms, where *K* is a parameter. To achieve greater diversity of selected terms across different rules (i.e. to avoid that the same top-quality terms be selected in all the rules where they appear) each term had its quality re-computed, by multiplying the original quality value by a random number in the range 0..1. Finally, the updated term quality value was then used to sort the terms using merge sort and only the top *K* terms were kept "turned on" in the pruned rule. The other terms were "turned off", by setting the corresponding attribute value to the state "off" in the particle representation.
2) Selecting a variable number of terms, in proportion to the quality of the terms. For instance, if the quality of a rule term (i.e., its normalised value in the range 0..1) was 0.6, then that term was preserved in the rule with a probability of 0.6, and therefore removed from the rule with the complementary probability of 0.4.

The above rule pruning procedures were alternatively used only to prune the rules resulting from the initial stage of seeding i.e. for the first iteration of the algorithm. Although only a number of rules equal to the size of the population need to be pruned, these rule pruning procedures were designed to be fast – at the price of ignoring term interactions, as mentioned above. This kind of fast pruning is important because right after seeding each rule is extremely long, having a number of terms equal to the total number of attributes in the data – since initially all terms are "turned on". However, at the end of the algorithm run, when the best rule is returned by the algorithm, it makes sense to prune that rule using a more sophisticated approach. At this point computational time is not a serious issue, since just one rule has to be pruned and the length of the rule should have been reduced by the rule optimisation process. Also the quality of the best rule found by the algorithm is a very serious issue, since this is a rule that will be used to classify new data later. Hence, we used the more sophisticated Ant-Miner's rule pruning procedure [12] for the final rule produced by each run of the hybrid PSO/ACO algorithm.

5 COMPUTATIONAL RESULTS

5.1 EXPERIMENTAL SETUP

The classes to be predicted in this project are 4 digit EC numbers (enzyme commission number), and the predictor attributes are Prosite [7] patterns – as discussed earlier. This data being mined (33079 examples, 854 attributes with boolean values representing the absence or presence of a ProSite pattern) was harvested from the UniProt [8] and Prosite databases. As a pre-processing step, classes with a number of records less than 10 were merged with their most similar sibling, since in principle there is not enough data to make a reliable prediction of those classes. The similarity between two classes was measured simply as the average number of matching attribute values between all records in either class. The total number of classes after this process was 850, with 6 classes at the first level, 52 at the second, 138 at the third and 654 at the fourth. During previous experiments it was found that, when using pruning

procedure (1), pruning the seeding rule to 50 terms produced the best results in terms of accuracy and comprehensibility and accuracy, and so this threshold value was used in the results reported in this paper.

5.2 RESULTS AND DISCUSSION

Table 1 reports the classification accuracy obtained by the proposed PSO/ACO and by our adapted version of the existing DPSO [11] – which was modified to use the same seeding and rule pruning procedure as the PSO/ACO algorithm. This adaptation was done in order to focus the comparison of the algorithms on the different ways that they represent and manipulate categorical values. DPSO was also adapted for hierarchical classification with the same top-down approach used by PSO/ACO. In addition, DPSO uses the same rule quality (particle fitness) formula as PSO/ACO, which is also the rule quality of Ant-Miner [12]. For each algorithm, Table 1 reports the accuracy of the two above-mentioned versions of the pruning procedure applied right after seeding, i.e. pruning preserving the top-50 terms and pruning preserving a variable number of terms. The classification accuracy on the test set (separated from training) was measured by a 5-fold cross-validation procedure [4], and the numbers after the symbol "±"denote standard deviations.

Table 1: Accuracy (%) of PSO/ACO and DPSO

	PSO/ACO		DPSO	
EC level	top-50 terms	var. No. of terms	top-50 terms	var. No. of terms
1	96.5±0.3	97.7±0.8	94.7±4.2	95.1±5.3
2	91.8±1.9	89.5±2.8	86.2±4.7	79.4±6.1
3	66.7±3.1	68.2±3.2	68.1±1.5	65.7±2.0
4	43.6±2.8	38.3±2.1	45.1±1.0	33.9±1.6

As can be seen from the results the accuracy of the predictions decreases with every level. This is expected for two reasons. First, the number of classes per level increases at deeper levels, with a corresponding decrease in the number of examples per class, making an accurate prediction at deeper levels more unlikely. Second, it is an inevitable result of using a divide and conquer type algorithm, as once an incorrect prediction has been made at a higher level it cannot be rectified, this leads to the accuracy being at best the same as the level above.

In any case, the classification accuracies at the first two levels are high. In particular, the accuracy at the first level – containing 6 classes – is above 96% for PSO/ACO. This is a very significant improvement over the baseline accuracy (the relative frequency of the majority class) at the first level. This baseline accuracy is the probability of predicting the correct class without using any classification algorithm, and rather just assigning the largest class to all examples in the test set. For the first level, the baseline accuracy is computed as the number of enzymes with class 2 (the majority class) in the training set, 7,764, divided by the total number of enzymes in the training set, 26,500, resulting in 29.3%. Although the accuracy values at the third and fourth level are relatively low, they are still much higher than the baseline accuracy for those levels, which are very low, given the very large number of classes at those deeper levels.

Let us now compare the results of PSO/ACO and DPSO in Table 1, for each kind of pruning. For the top-50 terms version of pruning, PSO/ACO outperformed DPSO at levels 1 and 2, whereas the opposite was true at levels 3 and 4. However, the differences in accuracies are not significant, taking into account the standard deviations.

For the pruning producing a variable number of terms, PSO/ACO outperformed DPSO at all levels, and the difference is significant at levels 2 and 4. This points towards the fact that PSO/ACO benefits more from the varied lengths of rules produced in that seeding process. The hybrid algorithm more quickly discards terms that none of the particles have found good, when compared to DPSO. This is demonstrated by the greater convergence times observed in the DPSO algorithm and the longer rules produced with DPSO, suggesting that a wider search is being performed. This may be a useful feature in other data sets but, due to the very large amount of attributes in the Enzyme data set, the more focused search of the PSO/ACO algorithm wins out.

Table 2: No. of rules generated per class level

	PSO/ACO		DPSO	
EC level	top-50 terms	var. No. of terms	top-50 terms	var. No. of terms
1	7.3± 0.2	8.0 ±0.0	9.2 ±0.6	8.0 ±0.0
2	54.3± 0.8	56.5 ±0.7	56.8 ±1.2	57.0 ±1.0
3	109.0± 0.9	112.3 ±0.7	110.3 ±1.1	110.7 ±1.3
4	477.0± 7.6	474.3 ±10.4	486.4 ±9.6	465.0 ±0.6

Table 3: Average No. of terms per rule per class level

	PSO/ACO		DPSO	
EC level	top-50 terms	var. No. of terms	top-50 terms	var. No. of terms
1	3.7±0.6	4.6±0.8	12.9±1.5	11.5±0.8
2	9.6±0.6	5.1±0.4	21.8±0.6	7.4±0.3
3	2.4±0.1	2.0±0.1	20.7±0.5	6.0±0.2
4	5.1±0.2	2.5±0.1	25.3±0.2	2.9±0.1

Let us now turn to the comprehensibility of the discovered rule set, which is measured by its size (number of rules and number of terms per rule), as usual in the data mining literature. Table 2 shows the number of rules generated by hybrid PSO/ACO and DPSO for each class level, whereas Table 3 shows the average number of terms per rule for these two algorithms per class level.

In general the number of rules discovered by PSO/ACO and DPSO is similar, for each kind of pruning. In addition, for each algorithm the kind of pruning used makes little difference in the number of rules. As expected, the number of rules increases at deeper levels of the class hierarchy, due to the corresponding increase in the number of classes. However, as shown in Table 3, the average number of terms in the rules discovered by PSO/ACO is much smaller than in the rules discovered by DPSO. To summarize, both algorithms discover about the same number of rules, but the rules discovered by PSO/ACO are much shorter, and so easier to be interpreted by the user.

6. CONCLUSIONS AND FUTURE RESEARCH

The paper has introduced a new hybrid PSO/ACO algorithm for hierarchical classification, and applied it to the classification of a challenging biological data set. The results were compared with an adapted version of an existing PSO algorithm. Overall the hybrid PSO/ACO obtained somewhat better results with respect to classification accuracy, and much better results with respect to comprehensibility of the discovered rule set.

Although in this paper the hybrid PSO/ACO was applied only to a biological data set, it is generic enough to be applied to other hierarchical classification data sets. So, a future research direction would be to evaluate the algorithm on other kinds of data sets. Also, the core idea of the hybrid algorithm, using an ACO-style pheromone-based mechanism for coping with categorical values, is independent of hierarchical classification, so it would be interesting to apply the algorithm even to other problems that do not involve data mining, but involve some kind of categorical (nominal, non-numeric) attributes. In addition, a lot of hierarchical data sets are also multi-label – i.e., there are several class attributes to be predicted. This scenario obviously add a lot of complexity and scope for optimisation to the problem in future research.

REFERENCES

[1] A. Sun and E.-P. Lim, *Hierarchical Text Classification and Evaluation*, Proc. 2001 IEEE ICDM (Int. Conf. on Data Mining), pp. 521-528, 2001

[2] H. Blockeel, M. Bruynooghe, S. Dzeroski, J. Ramon, and J. Struyf, *Hierarchical Multi-Classification*. SIGKDD Workshop on Multi-Relational Data Mining (MRDM-2002), pp. 21—35, 2002.

[3] M. Sasaki and K. Kita, *Rule-based text categorization using hierarchical categories*, In Proc. of the IEEE Int. Conf. On Systems, Man, and Cybernetics, pp. 2827-2830, 1998.

[4] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools with Java Implementations*, Morgan Kaufmann Publications, 2000.

[5] U.M. Fayyad, G. Piatetsky-Shapiro and P. Smyth. *From data mining to knowledge discovery: an overview*, Advances in Knowledge Discovery and Data Mining, AAAI/MIT, pp. 1-34, 1996.

[6] I. Shah L. Hunter, *Visualization based on the Enzyme Commission nomenclature*, Pacific Symposium on Biocomputing 3, pp. 142-152, 1998.

[7] ProSite, *http://us.expasy.org/prosite/*, Visited on Jan. 2005.

[8] UniProt, *http://www.ebi.uniprot.org/*, Visited on Jan. 2005.

[9] W. Tian1, A. K. Arakaki1 and J. Skolnick, EFICAz: a comprehensive approach for accurate *Genome-Scale Enzyme Function Inference*, Nucleic Acids Research, Vol. 32, No. 21, 2004.

[10] A. Ben-Hur and D. Brutlag, *Protein sequence motifs: Highly predictive features of protein function*, Technical Report, International Computer Science Institute, Berkeley, April 2004.

[11] T. Sousa, A. Silva, A. Neves, *Particle Swarm based Data Mining Algorithms for classification tasks*, Parallel Computing 30, pp. 767–783, 2004.

[12] R.S. Parpinelli, H.S. Lopes and A.A. Freitas. *Data Mining with an Ant Colony Optimization Algorithm*, IEEE Trans. on Evolutionary Computation, special issue on Ant Colony algorithms, 6(4), pp. 321-332, Aug. 2002.

[13] M. Dorigo and T. Stuetzle. *Ant Colony Optimization*. MIT Press, 2004.

[14] J. Kennedy and R. C. Eberhart, with Y. Shi. *Swarm Intelligence*, San Francisco: Morgan Kaufmann/ Academic Press, 2001.