

A Hierarchical Classification Ant Colony Algorithm for Predicting Gene Ontology Terms

Fernando E. B. Otero, Alex A. Freitas, and Colin G. Johnson

Computing Laboratory, University of Kent, Canterbury, UK
{febo2,A.A.Freitas,C.G.Johnson}@kent.ac.uk

Abstract. This paper proposes a novel Ant Colony Optimisation algorithm for the hierarchical problem of predicting protein functions using the Gene Ontology (GO). The GO structure represents a challenging case of hierarchical classification, since its terms are organised in a direct acyclic graph fashion where a term can have more than one parent — in contrast to only one parent in tree structures. The proposed method discovers an ordered list of classification rules which is able to predict all GO terms independently of their level. We have compared the proposed method against a baseline method, which consists of training classifiers for each GO terms individually, in five different ion-channel data sets and the results obtained are promising.

Keywords: Hierarchical classification, ant colony optimisation, protein function prediction.

1 Introduction

The large amount of uncharacterised protein data available for analysis has led to an increased interest in computational methods to support the investigation of the role of proteins in an organism. Protein classification schemes, such as the Gene Ontology [1] are organised in a hierarchical structure, allowing the annotation of protein at different levels of detail. In a hierarchical protein classification scheme, nodes near the root of the hierarchy represent more general functions while nodes near the leaves of the hierarchy represent more specific functions. The hierarchy also defines parent-child relationships between nodes where the child node is a specialisation of the parent node. From a data mining perspective, hierarchical classification is more challenging than single-level ‘flat classification’ [2]. Firstly, it is generally more difficult to discriminate between classes represented by leaf nodes than more general classes represented by internal nodes, since the number of examples per leaf node tends to be smaller compared to internal nodes. Secondly, an example may have more than one class predicted depending of its level in the class hierarchy and these predictions must satisfy hierarchical parent-child relationships.

This paper focuses on hierarchical protein function prediction using the Gene Ontology (GO) ‘molecular function’ domain. Note that the GO has a complex hierarchical organisation, where nodes are arranged in a directed acyclic graph

(DAG) structure and a particular node can have more than one parent — in contrast to only one parent in tree structures. We propose a new Ant Colony Optimisation (ACO) [3] classification algorithm, named *hAnt-Miner* (hierarchical classification Ant-Miner), for the hierarchical problem of protein function prediction using the GO structure. The proposed method discovers classification rules that predict functions at all levels of the GO hierarchy, and at the same time, which are consistent with the parent-child hierarchical relationships.

The remainder of this paper is organised as follows. Section 2 reviews related work. Section 3 describes the proposed hierarchical classification method using ACO. Section 4 describes the methodology used for the data preparation. Section 5 presents the computational results of the proposed method. Finally, Section 6 draws the conclusion of this paper and discuss future research directions.

2 Related Work on Gene Ontology Term Prediction

Much work on hierarchical classification of protein functions using the Gene Ontology has been focused on training a classifier for each GO term independently, using the GO hierarchy to determine positive and negative examples associated with each classifier [4], [5], [6]. Predicting each GO term individually has several disadvantages [7]. Firstly, it is slower since a classifier need to be trained n times (where n is the number of GO terms in the GO being predicted). Secondly, some GO terms could potentially have few positive examples in contrast to a much greater number of negative examples, particularly GO terms at deeper levels of the hierarchy. Many classifiers have problems with imbalanced class distributions [8]. Thirdly, individual predictions can lead to inconsistent hierarchical predictions, since parent-child relationships between GO terms are not imposed automatically during the training. However, more elaborate approaches can correct the individual predictions in order to satisfy hierarchical relationships — e.g. a Bayesian network is used to correct the inconsistent predictions of a set of SVM classifiers in [9]. Fourthly, the discovered knowledge identifies relationships between predictor attributes and each GO term individually, rather than relationships between predictor attributes and the GO hierarchy as a whole, which could give more insight about the data.

In order to avoid the aforementioned disadvantages of dealing with each GO term individually, a few authors have proposed classification methods that discover a single global model which is able to predict GO terms at any level of the hierarchy. Kiritchenko et al. [10] present an approach where the hierarchical problem is cast as a multi-label problem by expanding the label set (GO terms) of an example with all ancestor labels (ancestor GO terms). Then, a multi-label classifier is applied to the modified data set. For some examples, there is still need for a post-processing step to resolve inconsistencies in the GO terms predicted. Clare et al. [11] presented an adapted version of C4.5, which is able to deal with all GO term at the same time. They focused on discovering only a subset of very good rules for human analysis, rather than building a complete classification model for classifying the whole data set.

3 Proposed Method

The target problem of the proposed *hAnt-Miner* method is the discovery of hierarchical classification rules in the form *IF antecedent THEN consequent*. The antecedent of a rule is composed by a conjunction of predictor attribute conditions (e.g. `LENGTH > 25 AND IPR00023 = 'yes'`) while the consequent of a rule is composed by set of class labels (GO terms) in potentially different levels of the GO hierarchy (e.g. `GO:0005216, GO:0005244` — where `GO:0005244` is a subclass of `GO:0005216`). *IF-THEN* classification rules have the advantage of being intuitively comprehensible to biologists. *hAnt-Miner* divides the rule construction process into two different ant colonies, one colony for creating rule antecedents and one colony for creating rule consequents, which work in a cooperative fashion. Due to this paper's size restrictions this section assumes the reader is familiar with standard Ant Colony Optimisation (ACO) algorithms [3].

In order to discover a list of classification rules, a sequential covering approach is employed to cover all (or almost all) training examples. Algorithm 1 presents a high-level pseudo-code of the sequential covering procedure. The procedure starts with an empty rule list (*while* loop) and adds a new rule while the number of uncovered training examples is greater than a user-specified maximum value (*MaxUncoveredCases* parameter). At each iteration, a rule is created by an ACO procedure (*repeat-until* loop). Given that a rule is represented by trails in two different construction graphs, antecedent and consequent, two separated colonies are involved in the rule construction procedure. Ants in the antecedent colony create trails on the antecedent construction graph while ants in the consequent colony create trails on the consequent construction graph. In order to create a rule, an ant from the antecedent colony is paired with an ant from the consequent colony, so that the construction of a rule is synchronized between the two ant colonies. Therefore, it is a requirement that both colonies have the same number of ants (*ColonySize* parameter). The antecedent and

Algorithm 1. High level pseudo-code of the sequential covering procedure.

```

begin
  training_set ← all training examples;
  rule_list ← ∅;
  while |training_set| > max_uncovered_examples do
    rule_best ← ∅;
    i ← 1;
    repeat // ACO procedure
      rule_i ← CreateRule(); // use separate ant colonies for antecedent and
      consequent
      Prune(rule_i);
      UpdatePheromones(rule_i);
      if Q(rule_i) > Q(rule_best) then
        rule_best ← rule_i;
      end
      i ← i + 1;
    until i ≥ max_number_iterations OR rule convergence;
    rule_list ← rule_list + rule_best;
    training_set ← training_set - Covered(rule_best, training_set);
  end
end

```

consequent trails are created by probabilistically choosing a vertex to be added to the current trail (antecedent or consequent) based on the values of the amount of pheromone (τ) and a problem-dependent heuristic information (η) associated with vertices. There is a restriction that the antecedent of the rule must cover at least a user-defined minimum number of examples (*MinCasesPerRule* parameter), to avoid overfitting. Once the rule construction procedure has finished, the rule constructed by the ants is pruned to remove irrelevant terms from the rule antecedent and consequent. Then, pheromone levels are updated using a user-defined number of best rules (*UpdateSize* parameter) and the best-so-far rule is stored. The rule construction procedure is repeated until a user specified number of iterations has been reached (*MaxIterations* parameter), or the best-so-far rule is exactly the same in a predefined number of previous iterations (*ConvergenceTest* parameter). The best-so-far rule found, based on a quality measure Q , is added to the rule list and the covered training examples (examples that satisfy the rule's antecedent conditions) are removed from the training set.

The proposed *hAnt-Miner* is an extension of the 'flat classification' Ant-Miner [12] in several important ways, as follows. Firstly, it uses two separate ant colonies for constructing the antecedent and the consequent of a rule. Secondly, it uses a hierarchical classification rule evaluation measure to guide pheromone updating. Thirdly, it uses a new rule pruning procedure. Fourthly, it uses heuristic functions adapted for hierarchical classification.

3.1 Construction Graphs

Antecedent Construction Graph. Given a set of nominal attributes $\mathcal{X} = \{x_1, \dots, x_n\}$, where the domain of each nominal attribute x_i is a set of values $\mathcal{V}_i = \{v_{i1}, \dots, v_{im}\}$, and a set of continuous attributes $\mathcal{Y} = \{y_1, \dots, y_n\}$, the antecedent construction graph is defined as follows. For each nominal attribute x_i and value v_{ij} (where v_{ij} is the j -th value belonging to the domain of x_i) a vertex is added to the graph representing the term ($x_i = v_{ij}$). For each continuous attribute y_i a vertex is added to the graph representing the continuous attribute y_i . Since continuous attribute vertices do not represent a complete term (condition) to be added to a rule, when an ant visits a continuous attribute vertex, a threshold value is selected to create a term using ' $<$ ' or ' \geq ' relational operators (e.g. $y_i < value$). The selection of this value is deterministic and incorporates task-specific knowledge, increasing the effectiveness of the algorithm [13].

Then, vertices representing an attribute term (nominal or continuous) are connected to every other vertex referring to another attribute term, with the restriction that there are no edges between nominal attribute vertices referring to the same attribute (to avoid terms such as 'IPR00023 = yes' and 'IPR00023 = no' being included in the same rule). As a result, attribute term vertices are almost fully-connected. In addition, a dummy vertex '*start*' is added and unidirectionally connected to all vertices in the construction graph. This vertex represents the starting point for creating trails.

Consequent Construction Graph. Since the class labels are hierarchically structured as a directed acyclic graph (DAG), this structure can be directly used

to represent the consequent construction graph as follows. For each class label $l_i \in \mathcal{L}$, where \mathcal{L} is the hierarchy of class labels, a vertex is added to the graph. Subsequently, for every child vertex l_j of l_i where $l_j, l_i \in \mathcal{L}$, a directed connection from l_i to l_j is added to the graph. As a result, the consequent construction graph is a DAG, which is exactly the DAG of classes of the target problem, containing all classes and all parent-child class relationships in the target problem. Ants traverse the graph from the root vertex towards a leaf vertex. A created trail represents a set of predicted class labels, consistent with the hierarchy (satisfying parent-child relationships).

3.2 Rule Evaluation

Since the target problem of *hAnt-Miner* is the discovery of hierarchical classification rules, a variation of the hierarchical measure proposed in [10] is used to evaluate rules constructed by ants. The measure is a combination of both precision and recall hierarchical measures, and it takes into account the fact that an example belongs not only to its more specific class label, but also to all ancestor class labels according to the class hierarchy (except the root class label, since all examples trivially belong to the root class label by default).

As discussed earlier, the consequent of a rule is represented by a complete trail from the root class vertex to a leaf class vertex. In DAG structures, multiple paths between a given pair of class labels can exist. Therefore, immediately after an ant finishes building the consequent for rule r , the set of predicted class labels P_r of rule r is extended with the corresponding ancestor labels (P_r') as

$$P_r' = P_r \cup \{\cup_{l_i \in P_r} Ancestors(l_i)\} - l_{root}, \quad (1)$$

where $Ancestors(l_i)$ corresponds to all ancestor class labels of the class label l_i and l_{root} is the root class label of the hierarchy. The hierarchical macro-averaged measures of precision (hP) and recall (hR) are computed as

$$hP = \frac{\sum_{i \in S_r} \frac{|T_i \cap P_r'|}{|P_r'|}}{|S_r|} \quad hR = \frac{\sum_{i \in S_r} \frac{|T_i \cap P_r'|}{|T_i|}}{|S_r|}, \quad (2)$$

where S_r is the set of all examples covered by (satisfying the rule antecedent of) rule r and T_i is the set of true class labels of the i -th example. The hierarchical precision (hP) is the average number of true class labels that are predicted by rule r divided by the total number of predicted class labels across the examples covered by rule r . The hierarchical recall (hR) is the average number of true class labels that are predicted by rule r across the examples covered by rule r divided by the total number of true class labels which should have been predicted across the examples covered by rule r .

The rule quality measure Q is defined as a combination of the hP and hR measures, equivalent to the hierarchical F-measure, given by Equation (3)

$$Q = hF = \frac{2 \cdot hP \cdot hR}{hP + hR}. \quad (3)$$

3.3 Rule Pruning

The rule pruning procedure aims at improving the rule quality by removing irrelevant terms that might have been added during the rule construction process and it is applied as soon as the rule construction is completed. Recall that a rule is composed by antecedent and consequent parts, which in turn are represented by different ant trails that might contain irrelevant vertices.

A rule undergoes the pruning procedure as follows. At the first step, the quality of the rule is computed using the quality measure Q as given by Equation (3). In the second step, the rule is submitted to an iterative removal of the last term added to its antecedent while the quality of the rule is improved. At each iteration, the consequent of a candidate rule is also submitted to an iterative removal of the last added class label in an attempt to improve the generalization behaviour of the candidate rule. Note that, for the purpose of both these iterative removal procedures, the terms and classes (in the antecedent and consequent, respectively) are considered as an ordered list, and terms and classes are removed in an order inverse to the order in which they were added to the rule.

Algorithm 2 describes the rule pruning. Let $rule_r$ be the rule undergoing the pruning procedure and q_r be the quality measure of $rule_r$. At each iteration of the outer repeat loop in Algorithm 2, a candidate rule $rule_i$ is created by removing the last term of the antecedent of $rule_r$ and its quality measure q_i is computed. Subsequently, j ($0 < j < |rule_i.consequent|$) candidate rules are sequentially created by removing the last j class label(s) of the consequent of $rule_i$. This is implemented by the inner repeat loop in Algorithm 2. If the quality measure of a $rule_j$ is higher than q_i , $rule_i$ is substituted by $rule_j$. Finally, $rule_i$ substitutes $rule_r$ if $q_r \leq q_i$, completing an iteration of the pruning procedure. This procedure is repeated until $rule_r$ has just one term left on its antecedent or a candidate rule $rule_i$ does not improve the quality over $rule_r$ (i.e. $q_r > q_i$).

Algorithm 2. Rule pruning procedure pseudo-code.

```

begin
  rulebest ← rule;
  qbest ← Q(rulebest);
  repeat
    antecedent ← rulebest.antecedent – last_term(rulebest.antecedent);
    rulei ← antecedent + rulebest.consequent;
    qi ← Q(rulei);
    consequentj ← rulebest.consequent;
    repeat
      consequentj ← consequentj – last_class(consequentj);
      rulej ← antecedent + consequentj;
      if (Q(rulej) > qi) then
        rulei ← rulej;
        qi ← Q(rulej);
      end
    until |consequentj| = 1 ;
    if (qi ≥ qbest) then
      rulebest ← rulei;
      qbest ← qi;
    end
  until qi < qbest OR |rulebest.antecedent| = 1 ;
end

```

3.4 Pheromone Trails

Pheromone Initialisation. In order to reinforce trails followed by ants that constructed good rules, pheromone values are associated with edges in the antecedent and consequent construction graphs. For each vertex i of both antecedent and consequent construction graphs, the initial amount of pheromone deposited at each edge is inversely proportional to the number of edges originating at vertex i , computed as

$$\tau_{edge_{ij}}(t = 0) = \frac{1}{|E_i|}, \quad (4)$$

where E_i is the set of edges originating at vertex i , $edge_{ij}$ is the edge that connects vertex i to its j -th neighbour vertex and t is the time index. As a result of Equation (4), the same amount of pheromone is initially associated with every $edge_{ij}$ coming out from vertex i .

Pheromone Updating. The pheromone trails followed by ants are updated based on the quality of the rule that they represent, which in turn guides future ants towards better trails. Since a rule is composed by antecedent and consequent trails, the pheromone update procedure is divided in two steps.

In the first step, the trail that represents the antecedent of a rule r is updated as follows. Starting from the dummy ‘start’ vertex (0-th vertex), the pheromone value of the edge that connects the i -th vertex to the $(i + 1)$ -th vertex ($0 \leq i < |rule.ancecedent|$) is incremented according to

$$\tau_{edge_{ij}}(t + 1) = \tau_{edge_{ij}}(t) + \tau_{edge_{ij}}(t) \cdot q_r, \quad (5)$$

where i and j are the i -th and j -th vertices of an edge from i to j in the trail being updated ($edge_{ij}$) and q_r is the quality measure of rule r — Equation (3).

In the second step, the pheromone value of every edge of the consequent of rule r that connects the i -th vertex to the $(i+1)$ -th vertex ($0 < i < |rule.consequent|$) is incremented according to Equation (5). Note that, before computing the rule quality, the consequent is expanded to include all ancestor class labels of the class labels originally added to the rule’s consequent by an ant, since there can be multiple paths between class labels, as detailed in subsection 3.2. However, during pheromone updating, only the actual trail that was followed to create the original consequent is updated. This avoids reinforcing trails that did not directly contribute to the consequent construction.

Pheromone Evaporation. This is implemented by normalizing the pheromone values of edges of each construction graph G (antecedent and consequent). The normalization procedure indirectly decreases the pheromone of unused edges, since the pheromone of used edges has been increased by Equation (5). This normalization is given by

$$\tau_{edge_{ij}} = \frac{\tau_{edge_{ij}}}{\sum_{\tau_{edge_{ij}} \in G} \tau_{edge_{ij}}}. \quad (6)$$

3.5 Heuristic Functions

Antecedent Heuristic Function. The heuristic function used in the antecedent construction graph is based on information theory, more specifically, it involves a measure of the entropy associated with each term (vertex) of the graph. In the case of nominal attributes, where a term has the form $(x_i = v_{ij})$, the entropy for the term is computed as

$$entropy(term_{x_i=v_{ij}}) = \sum_{k=1}^{|\mathcal{L}|} -p(l_k | term_{x_i=v_{ij}}) \cdot \log_2 p(l_k | term_{x_i=v_{ij}}), \quad (7)$$

where $p(l_k | term_{(x_i=v_{ij})})$ is the empirical probability of observing class label l_k conditional on having observed $x_i = v_{ij}$ (attribute x_i having the specific value v_{ij}) and $|\mathcal{L}|$ is the total number of class labels. The entropy is a measure of the impurity in a collection of examples, hence higher entropy values correspond to more uniformly distributed classes and smaller predictive power for the term in question. Equation (7) is a direct extension of the heuristic function of the original Ant-Miner [12] (for ‘flat classification’) into the problem of hierarchical classification.

In the case of continuous attributes, where a vertex represents just an attribute (and not an attribute-value pair), a threshold value v is chosen to dynamically partition the continuous attribute y_i into two intervals: $y_i < v$ and $y_i \geq v$. *hAnt-Miner* chooses the threshold value v that minimizes the entropy of the partition, given by

$$entropy(y_i, v) = \frac{|S_{y_i < v}|}{|S|} \cdot entropy(y_i < v) + \frac{|S_{y_i \geq v}|}{|S|} \cdot entropy(y_i \geq v), \quad (8)$$

where $|S_{y_i < v}|$ is the total number of examples in the partition $y_i < v$ (partition of training examples where the attribute y_i has a value less than v), $|S_{y_i \geq v}|$ is the total number of examples in the partition $y_i \geq v$, $|S|$ is the total number of training examples, and $entropy(y_i < v)$ and $entropy(y_i \geq v)$ are the entropy of the terms represented by $(y_i < v)$ and $(y_i \geq v)$ as given by Equation (7).

After the selection of the threshold v_{best} , the entropy of the term representing the continuous attribute y_i corresponds to the minimum entropy value of the two partitions and it is defined as

$$entropy(term_{y_i}) = \min(entropy(y_i < v_{best}), entropy(y_i \geq v_{best})). \quad (9)$$

Equations (8) and (9) are derived from the Ant-Miner version for coping with continuous attributes, described in [13]. In this current paper the heuristic function is straightforwardly extended to hierarchical classification as follows. Since the entropy of the i th-term (nominal or continuous) of the antecedent construction graph varies in the range $0 \leq entropy(term_i) \leq \log_2(|\mathcal{L}| - 1)$ (where $|\mathcal{L}| - 1$ is the number of class labels in the class hierarchy without considering the root class label) and lower entropy values are preferred over higher values, the heuristic function is computed as

$$\eta_{term_i} = \log_2(|\mathcal{L}| - 1) - entropy(term_i), \quad \forall term_i \in G_A, \quad (10)$$

where $term_i$ is the i th-term of the antecedent construction graph G_A . Equation (10) will give a higher probability of being selected to terms with lower entropy values, which corresponds to terms with higher predictive power.

Consequent Heuristic Function. The heuristic function used in the consequent construction graph is based on the frequency of training examples for each class label of the hierarchy, given by

$$\eta_i = |TR_i|, \quad \forall l_i \in G_C, \quad (11)$$

where $|TR_i|$ is the number of training examples that belong to class label l_i and G_C is consequent construction graph. Note that the heuristic function has a bias towards class labels that have a greater number of examples, which therefore will initially favour the discovery of rules with these class labels in the consequent. However, due to the use of a sequential covering procedure, rules predicting less frequent classes will be eventually discovered as well.

4 Bioinformatics Data Preparation

In order to evaluate the proposed method, we have created five data sets involving ion-channel protein functions. Ion channel proteins are present in all living cells and they form a pore across the cell membrane [14]. The function of ion channels is to allow specific inorganic ions (e.g. Na^+ , K^+ , Ca^{2+} , Cl^-) to cross the cell membrane. They play an essential role in many cell functions, such as in functions related to the nervous system, muscle contraction and T-cell activation.

The selection of the protein examples was divided into three steps. In the first step we selected a subset of the Gene Ontology hierarchy to represent the hierarchical classes to be predicted. As we focus on ion channel proteins, all the ancestor and descendant terms of the GO:0005216 (*ion channel activity*) term were selected. In the second step, we retrieved protein interaction data from the IntAct database (release 15/12/2007). Records with database cross-references to the GO terms selected in the previous step were retrieved. Since many GO terms (classes) selected in the previous step did not have a reasonable number of proteins associated with them, we discarded GO terms with less than 10 protein examples. In the third step, for each protein example retrieved in the previous step we selected the amino acid sequence and InterPro pattern references from the UniProt database (release 12.0), using the database cross-reference to UniProt found in IntAct protein records. We ended up with 147 protein examples involving 17 GO terms, which were used to create three different data sets. The first data set ('DS1 AA') used the amino acid composition information as predictor attributes, consisting of the percentage of the sequence composition relative to each of the 20 different amino acids. The second data set ('DS1 InterPro') used the InterPro pattern information as predictor attributes, consisting of boolean attributes representing the presence or absence of a particular InterPro pattern. The third data set ('DS1 IntAct') used the IntAct protein interaction data as predictor attributes, consisting of boolean attributes representing the presence or absence of a particular interaction.

Using a similar approach, without the restriction of selecting proteins with protein interaction data available, we increased the number of proteins to 359 examples to create two additional data sets. The first data set ('DS2 AA') used the amino acid composition information as predictor attributes. The second data set ('DS2 InterPro') used the InterPro pattern information as predictor attributes.

5 Computational Results

The proposed *hAnt-Miner* method was compared against a baseline approach, which consists of training a classifier for each GO term of the hierarchy individually. The J48 classifier (Weka [15] implementation of the well-known C4.5 decision tree algorithm [16]) was chosen in this approach. In this case, the GO hierarchy was used to determine the set of positive/negative examples for every individual classifier as follows. For each classifier associated with a particular GO term (the classifier which predicts if an example belongs or not to the particular GO term), the set of positive examples consists of all examples that belong to the GO term in question (as the most specific GO term as an ancestor of their most specific GO term); the set of negative examples consists of all the remaining training examples. After training the individual classifiers, a test example is classified in a top-down fashion. First, the example is classified only by the child classifiers of the root GO term. For each child classifier, if the classifier predicts the positive class (the example is predicted to belong to the GO term associated with the classifier), then the example is 'pushed downwards' and classified by its children classifiers. This procedure goes on until a classifier does not predict the GO term (the example is predicted as a negative example) or when a leaf classifier is reached. At the end of this procedure, the set of predicted class labels is consistent with the GO hierarchy.

We compare the performance of *hAnt-Miner* and J48 in terms of the hierarchical measures of precision, recall and F-measure, since standard classification accuracy measures are not suitable for hierarchical classification (i.e. they do not account for misclassification errors at different levels of the hierarchy). The hierarchical measures of precision, recall and F-measure are defined in Equations (2) and (3) respectively, with the difference that all test examples are considered as we are evaluating a classification model and not a rule (which is the case in subsection 3.2). The experiments were conducted running the well-known 10-fold cross-validation procedure [15] and the results are reported as average values with standard deviation computed over the 10 different iterations. In all experiments, the parameters of *hAnt-Miner* were set to: '*ColonySize* = 20', '*MinCasesPerRule* = 5', '*MaxUncoveredCases* = 10', '*ConvergenceTest* = 10', '*MaxIterations* = 500' and '*UpdateSize* = 1'. We have made no attempt to optimise these parameters for the data sets used in the experiments. The results comparing the proposed *hAnt-Miner* method against J48 are shown in Table 1.

Overall *hAnt-Miner* achieved better results than J48 in our set of experiments. J48 was significantly outperformed (according to a Student's t-test — see Table 1) in two out of five data sets, namely 'DS1 InterPro' and 'DS1 IntAct'. These data sets can be considered 'difficult' based on their small size (147 proteins)

Table 1. Hierarchical measures of precision (hP), recall (hR) and F-measure (hF) values (*mean \pm standard deviation*) obtained with the 10-fold cross-validation procedure in the five data sets. An entry in the ‘hF’ column is shown in bold if the hierarchical F-measure value obtained by one of the methods was significantly greater than the other method — according to a two-tailed Student’s t-test with 99% confidence.

	hP	J48 (top-down) hR	hF
DS1 AA	0.73 \pm 0.04	0.55 \pm 0.03	0.63 \pm 0.03
DS1 InterPro	0.69 \pm 0.04	0.68 \pm 0.05	0.69 \pm 0.04
DS1 IntAct	0.69 \pm 0.03	0.37 \pm 0.04	0.47 \pm 0.03
DS2 AA	0.71 \pm 0.02	0.61 \pm 0.02	0.65 \pm 0.02
DS2 InterPro	0.91 \pm 0.01	0.84 \pm 0.02	0.87 \pm 0.01
	hP	<i>hAnt-Miner</i> hR	hF
DS1 AA	0.56 \pm 0.06	0.55 \pm 0.06	0.56 \pm 0.06
DS1 InterPro	0.82 \pm 0.04	0.81 \pm 0.04	0.81 \pm 0.04
DS1 IntAct	0.77 \pm 0.04	0.54 \pm 0.03	0.63 \pm 0.03
DS2 AA	0.63 \pm 0.02	0.59 \pm 0.02	0.61 \pm 0.01
DS2 InterPro	0.83 \pm 0.01	0.75 \pm 0.01	0.79 \pm 0.01

and distribution of examples in the GO hierarchy (GO terms at deeper levels of the hierarchy have few examples). Therefore, the poor performance of J48 could be the result of the problem that there are many more negative examples than positive examples for each GO node, particularly at deeper level in the GO hierarchy, which shows that *hAnt-Miner* is more robust than J48 when dealing with unbalanced hierarchical class distributions. This problem was not observed in the experiments concerning the data sets with a greater number of protein examples, where J48 significantly outperformed *hAnt-Miner* in the ‘DS2 InterPro’ data set. In the remaining two data sets, namely ‘DS1 AA’ and ‘DS2 AA’, there were no significant difference between both methods.

6 Conclusion

This paper has presented a new Ant Colony Optimisation algorithm, named *hAnt-Miner*, for the hierarchical classification problem of predicting protein functions using the Gene Ontology (GO). The proposed *hAnt-Miner* discovers a single global classification model in the form of an ordered list of *IF-THEN* classification rules which can predict GO terms at all levels of the GO hierarchy, satisfying the parent-child relationships between GO terms. Experiments comparing *hAnt-Miner* with a baseline method based on J48, where one classifier is trained individually for each GO term of the hierarchy, have shown positive results: *hAnt-Miner* was significantly more accurate than J48 in two (out of five) data sets, with the reverse being true in just one data set.

There are several possible avenues for future research. It would be interesting to investigate different rule evaluation measures in order to optimise the quality

of constructed rules. Different variations of the rule pruning procedure could prove to be more effective. Producing an unordered rule set instead of an ordered rule list could give more flexibility to the algorithm. Finally, it would be interesting to apply *hAnt-Miner* to more bioinformatics data sets.

Acknowledgements. The authors acknowledge the financial support from an European Union's INTERREG project (Ref. No. 162/025/361). Fernando Otero also acknowledges further financial support from the Computing Laboratory, University of Kent.

References

1. The Gene Ontology Consortium: Gene Ontology: tool for the unification of biology. *Nature Genetics* 25, 25–29 (2000)
2. Freitas, A., de Carvalho, A.: A tutorial on hierarchical classification with applications in bioinformatics. In: Taniar, D. (ed.) *Research and Trends in Data Mining Technologies and Applications*, pp. 175–208. Idea Group, USA (2007)
3. Dorigo, M., Stützle, T.: *Ant Colony Optimization*. MIT Press, Cambridge (2004)
4. Jensen, L., Gupta, R., Stærfeldt, H., Brunak, S.: Prediction of human protein function according to Gene Ontology categories. *Bioinformatics* 19(5), 635–642 (2003)
5. Lægread, A., Hvidsten, T., Midelfart, H., Komorowski, J., Sandvik, A.: Predicting gene ontology biological process from temporal gene expression patterns. *Genome Research* 13(5), 965–979 (2003)
6. Bi, R., Zhou, Y., Lu, F., Wang, W.: Predicting Gene Ontology functions based on support vector machines and statistical significance estimation. *Neurocomputing* 70, 718–725 (2007)
7. Blockeel, H., Schietgat, L., Struyf, J., Džeroski, S., Clare, A.: Decision Trees for Hierarchical Multilabel Classification: A Case Study in Functional Genomics. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) *PKDD 2006. LNCS (LNAI)*, vol. 4213, pp. 18–29. Springer, Heidelberg (2006)
8. Japkowicz, N., Stephen, S.: The class imbalance problem: a systematic study. *Intelligent Data Analysis* 6, 429–450 (2002)
9. Barutcuoglu, Z., Schapire, R.E., Troyanskaya, O.G.: Hierarchical multi-label prediction of gene function. *Bioinformatics* 22(7), 830–836 (2006)
10. Kiritchenko, S., Matwin, S., Famili, A.F.: Functional annotation of genes using hierarchical text categorization. In: *BioLINK SIG: Linking Literature, Information and Knowledge for Biology* (2005)
11. Clare, A., Karwath, A., Ougham, H., King, R.: Functional bioinformatics for *Arabidopsis thaliana*. *Bioinformatics* 22(9), 1130–1136 (2006)
12. Parpinelli, R., Lopes, H., Freitas, A.: Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation* 6(4), 321–332 (2002)
13. Otero, F., Freitas, A., Johnson, C.: *cAnt-Miner*: an ant colony classification algorithm to cope with continuous attributes. In: Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A.F.T. (eds.) *ANTS 2008. LNCS*, vol. 5217, pp. 48–59. Springer, Heidelberg (2008)
14. Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P.: *The Molecular Biology of the Cell*, 4th edn. Garland Press (2002)
15. Witten, H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn. Morgan Kaufmann, San Francisco (2005)
16. Quinlan, J.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco (1993)